cégep GÉRALD GODIN

Cours 7

Les fils d'exécution (threads)

Synchronisation et communication

Qu’est-ce qu’un thread

Un thread est un sous-processus pouvant s'exécuter en parallèle avec d'autres threads et pouvant utiliser des ressources communes

Nous utilisons intensivement les threads lors de l'exécution d'une classe : **le *Garbage Collector*** qui vient nettoyer la mémoire s'exécute par exemple au sein d'un thread séparé

Création d'un thread

Dériver la classe de *java.lang.Thread*. Cette classe possédera une méthode ***run()*** qui contient le code principal du thread.

Implémenter l'interface *Runnable*.Cette classe implementera l'unique méthode ***run()*.**

Dériver la classe Thread

```
public class C1 extends Thread {

public void run() {
....
}

public static void main(String[] args) {
C1 objC1 = new C1();
objC1.start();
}

}
```

Dériver la classe Thread

```
public class C1 extends Thread {

public C1() {
....
start();
}
 public void run() {
....
 }
public static void main(String[] args) {
C1 objC1 = new C1();   // crée et démarre le thread
}

}
```

Implémenter la classe Runnable

```
public class C2 implements Runnable {

public void run()    { // méthode de l'interface Runnable
...

}

public static void main(String[] args) {
C2 objC2 = new C2();
Thread proc = new Thread( objC2);  // crée le thread
proc.start(); // le démarre en appelant proc.run
}
}
```

Implémenter la classe Runnable

```
public class C2 implements Runnable {

public C2() {
...
 new Thread(this).start();   // crée et démarre le thread
}

public void run() { // méthode de l'interface Runnable
...
}
public static void main(String[] args) {
C2 objC2 = new C2();   // crée l'objet et démarre le thread
}
}
```

La classe Thread

La classe *Thread* encapsule les moyens de contrôle des fils.

L'objet *Thread* n'est pas un fil, mais plutôt une sorte de télécommande de fil, le lien qui permet de gérer le comportement des fils. Des méthodes permettent d'exécuter le fil, de le faire dormir ou de l'interrompre.

Avant de pouvoir manipuler un fil, il faut en obtenir une référence. Une façon de faire consiste à utiliser la méthode *static currentThread* qui permet de récupérer le fil en cours d'exécution.

Exemple 1

faire dormir le fil courant et lui changer de nom

Exemple 1

- Le programme affiche, après 4 secondes : Le fil principal est Thread[Fil principal,5,main].

- "Fil principal" est le nom du fil, 5 sa priorité et main son groupe de fils (le groupe permet de contrôler l'état d'un ensemble de fils).

- Notez que la méthode static sleep s'applique au fil courant et n'a pas besoin d'obtenir une instance référant à ce fil. La méthode sleep peut lancer une exception de type InterruptedException si un autre fil a interrompu celui qui dort.

L’interface Runnable

Pour travailler avec plusieurs fils, il suffit de lancer un nouveau fil sur n’importe quel objet qui implante *Runnable*, une interface simple qui définit abstraitement l’exécution asynchrone d’un code.

Une classe peut implanter *Runnable* en définissant une seule méthode : *run*.

Dans l’exemple suivant, on démarre un fil nommé "Second Fil", qui affiche à chaque seconde un nombre en commençant à 1 et en se terminant à 5. Dès que le second fil est démarré, le fil principal attend 3 secondes, puis affiche un message de sortie et termine.

Affichage du programme

```
Fil principal : Thread[main,5,main]
Second fil : Thread[Second Fil,5,main]
1
2
3
Fin du fil principal
4
5
Fin du second fil.
```

Thread JavaFX Application

-Fil d'exécution en tâche de fond, qui gère les événements liés à l'interface graphique (rafraîchissement de composants, événement de souris, de clavier,…)

-Voir démontration

Concurrence d’accès

Des"verrous" peuvent être posés sur des blocs d'instruction ou des méthodes d'une classe afin d'éviter les états indéterminés en cas d'accès concurrents. Il faut poser le mot-clé ***synchronized*** sur une méthode ou un bloc d'instructions qui accède aux données susceptibles d'être modifiées par plusieurs threads en même temps

Concurrence d'accès…

Lorsqu'un fil se trouve à l'intérieur d'une méthode ***synchronized***, aucun autre fil ne peut appeler une autre méthode ***synchronized*** sur le même objet. On dit alors que « le moniteur est verrouillé » ou que « le fil est entré dans le moniteur ».

sleep, wait et notify

un thread peut se mettre en attente d'un événement par la méthode *wait()* qui doit être appelée dans la classe même. Dans ce cas, le thread attendra une notification de reprise d'activité pour reprendre. Un appel de ***notify()*** (classe *Object*) sur une instance de Thread permet de notifier à ce dernier de reprendre son exécution bloquée par un *wait()*. Un ***notifyAll()*** permet cette action sur tous les threads en attente.

Attention: Les méthodes *sleep()* et *wait()* peuvent provoquer des *InterruptedException*.

Exemple de synchronisation

Une classe **Messager** permet d'afficher un message à l'aide de sa méthode **afficherMessage()** La méthode attend 3 secondes puis affiche le message demandé.

La classe **TestSynchronisation** crée une instance de la classe Messager qu'elle passe à trois instances de la classe de **AppelDeMessager.** Noter bien ce qui se passe puisque nous synchronisons les appels.

Affichage avec synchronisation

On entre dans la méthode afficherMessage

Message 1

On entre dans la méthode afficherMessage

Message 2

On entre dans la méthodeafficherMessage

Message 3

Affichage sans synchronisation

On entre dans la méthode afficherMessage
On entre dans la méthode afficherMessage
On entre dans la méthode afficherMessage
Message 3
Message 2
Message 1

Communication entre les processus

wait, notity et notifyAll sont les mécanismes de communication entre processus offert par Java. Ces méthodes sont implantées dans le classe Object avec le modificateur final (pourquoi?).

Ne pas oublier de déclarer synchronized, les méthodes qui appellent wait, notify et notifyAll

producteur/consommateur (capacité du tampon = 1)

Nous avons une classe **Producteur** qui produit des produits et une classe **Consommateur** qui consomme les ptroduits. Un produit créé doit avoir un numéro unique et aucun produit ne peut être produit tant qu'un consommateur n'a pas consommé le dernier produit disponible.

La classe **Tampon** représente l'objet que l'on veut synchroniser

wait, notify, notifyAll

wait: le fil en cours d'exécution cède le moniteur et dort jusqu'à ce qu'un autre fil entre dans le moniteur et appelle notify

notify: réveille un des fils qui a appelé wait sur le même objet

notifyAll : réveille tous les fils qui ont appelé wait sur le même objet.

Principales méthodes de la classe Thread

static Thread currentThread()

static void sleep(long millisecondes)

static yield(): le processus en cours cesse temporairement de s'exécuter.

String getName()

void setName(String nom)

int getPriority()

int setPririority

Boolean isAlive(): vérifie si le fil est en vie..