Exercices Cours 16

Etude de l’interruption.

Exercice 1 : Reprendre la classe *BonjourPote* du cours 15. Utiliser comme condition d’arrêt dans la méthode *run* la méthode *isInterrupted* de la classe *Thread*. Puis tester avec la méthode suivante :

**private** **void** testInterruption()

{

BonjourPote3 bonjour = **new** BonjourPote3("Jules") ;

bonjour.lancer() ;

**try**{Thread.*sleep*(3000);} **catch**(InterruptedException e){}

bonjour.getThreadCourant().interrupt() ;

System.*out*.println("JVM : Interruption");

}

Que constatez-vous ?

Exercice 2 : Lancer le même test après avoir supprimé ou mis en commentaire la ligne de temporisation : **try**{Thread.*sleep*(500);} **catch**(InterruptedException e){} dans la méthode *run*.

Que constatez-vous ? A quoi attribuez-vous la différence ?

Exercice 3 : Même tests que l’exercice 2, mais ajouter les lignes de code en fin de boucle dans la méthode *run* :

interruption = threadCourant.isInterrupted() ? "est interrompu" : "non interrompu" ;

**if** (interruption.startsWith("est"))

sauvegarder(interruption) ;

et la méthode sauvegarder :

**private** **void** sauvegarder(String interruption)

{

File fich = **new** File("save/interruption.txt") ;

fich.getParentFile().mkdirs() ;

**try**

{

PrintStream sortie = **new** PrintStream(**new** FileOutputStream(fich)) ;

sortie.println(interruption) ;

sortie.close() ;

} **catch** (FileNotFoundException e){}

}

Lancer plusieurs fois le test, en remplaçant parfois *isInterrupted*() par *interrupted*() dans l’affectation de la variable *interruption*. Que constatez-vous ? Quelles précautions cela vous incite-t-il à prendre ?

Etude de la méthode *join*.

Exercice 4 : reprendre la classe *BonjourPote*, et modifier la comme suit :

* Ajouter un attribut *valeurMax* entier.
* Dans le constructeur, affecter cet attribut d’une valeur aléatoire comprise entre 0 et 20.
* La boucle de la méthode *run* doit s’arrêter uniquement lorsque le compteur a atteint *valeurMax*. Il faut une temporisation dans la boucle.

Ecrire ensuite une méthode de test qui crée trois objets de type *BonjourPote*, les fait exécuter simultanément dans des processus threads distincts. Puis attend la fin des trois processus avant d’afficher que les trois sont terminés.

Lancer le test plusieurs fois. Le lancer aussi sans l’attente.

Exercice 5 : Reprendre le test précédent en ajoutant dans l’affichage de la boucle de la méthode *run* de la classe *BonjourPote* le nom du thread courant. Avec ou sans modification de ce nom.

Concurrence et situations critiques

Exercice 6 :

1. Créer une classe *Banque* qui aura pour seul attribut une map de soldes (des entiers) indexés sur le nom du titulaire du compte. Le constructeur l’initialisera avec le compte de « Dupont » qui contiendra 1000 euros. Lui adjoindre les accesseurs *getSolde*(*nomDuTitulaire*) et *setSolde*(*nomDuTitulaire, solde*). On pourra, pour simplifier, en faire un singleton.
2. Créer une classe *Distributeur* ayant pour attribut la banque, et un entier *deltaTemps* initialisé à 100. Cette classe contiendra deux méthodes : *int retirer*(*int montant, Client client*) et pour *void deposer*(*int montant, Client client*).
3. Créer une classe *Client* qui possède un nom de titulaire, un prénom, un montant à retirer, et qui soit *Runnable* qu’on puisse lancer et arrêter à volonté. L’action du client sera de déposer ou retirer 10 fois de suite le montant (suivant qu’il est positif ou négatif), à un distributeur.
4. Le test consistera à récupérer la banque, patienter une demi-seconde (question de mise en place du singleton), puis créer et lancer deux clients : Dupont Paul pour les dépôts de 100 euros et Dupont Virginie pour des retraits de 100 euros.

La méthode *retirer* du distributeur consiste à récupérer les nom et prénom du client paramètre, demander le solde à la banque, lui retirer le montant demandé si le solde est suffisant, patienter le *deltaTemps* en attribut, puis afficher le prénom du client, le montant effectivement retiré et le solde, enfin indiquer à la banque le nouveau solde et retourner le montant retiré.

La méthode *déposer* du distributeur consiste à récupérer les nom et prénom du client paramètre, demander le solde à la banque, lui ajouter le montant paramètre, patienter le *deltaTemps* en attribut, puis afficher le prénom du client, le montant déposé et le solde, enfin indiquer à la banque le nouveau solde.

Que constatez-vous après plusieurs exécutions du test ? A quoi cela est dû ?

Exercice 7 : Corriger les failles de la question précédente.

Exercice 8 : Dans les méthodes *lancer* et *arrêter* des classe *BonjourPote*, il y a un problème : on peut lancer plusieurs processus identiques :

Exercice 9 : Reprendre un TP sur les réservations de places pour un spectacle. Considérer des réservations concurrentes. Déterminer les situations critiques et y remédier. Programmer une simulation par une arrivée aléatoire de spectateurs qui réservent des places. (Les simulateurs seront des processus indépendants et arriveront à des bureaux de réservation distincts).

Exercice 10 (le TP) : Reprendre le TP sur les robots et faire les pièces doivent être indépendantes (s’exécuter dans des fils d’exécutions distincts).