PROGRAMMATION EN JAVA

Semaine 9: Programmation multithread

Enseignant: NGUYỄN Thị Minh Tuyền

Plan du cours

1. Fondamentaux du multithreading
2. Synchronisation

Plan du cours

1. Fondamentaux du multithreading
2. Synchronisation

Multitâche

- Deux types de multitâche:
  - basé sur les processus et
  - basé sur les threads.
- Le multitâche basé sur les processus est la fonctionnalité qui permet à votre ordinateur d'exécuter deux programmes ou plus simultanément.
  - Un programme est la plus petite unité de code pouvant être envoyée par le planificateur.
- Dans un environnement multitâche basé sur des threads:
  - Le thread est la plus petite unité de code distribuable.
  - Un seul programme peut effectuer deux tâches ou plus à la fois.
- Le multitâche basé sur les processus n'est pas sous le contrôle de Java. Le multitâche multithread est.

Avantage du multithreading

- Il vous permet d'utiliser le temps d'inactivité présent dans la plupart des programmes.
  - Un programme passera souvent la majorité de son temps d'exécution à attendre d'envoyer ou de recevoir des informations vers ou depuis un appareil.
  - En utilisant le multithreading, votre programme peut exécuter une autre tâche pendant cette période d'inactivité.
- Les fonctionnalités de multithreading de Java fonctionnent à la fois dans les systèmes monocœur et multicœurs.
  - Dans un système monocœur: les threads exécutant simultanément partagent le CPU, chaque thread recevant une tranche de temps CPU - deux threads ou plus ne s'exécutent pas en même temps, mais le temps CPU inactif est utilisé.
  - Dans les systèmes multiprocesseurs / multicœurs: deux threads ou plus peuvent être exécutés simultanément.

Thread

- États d'un thread:
  - en cours d'exécution,
  - suspendu,
  - repris,
  - bloqué,
  - terminé.
- La synchronisation permet de coordonner l'exécution des threads de certaines manières bien définies.

Classe Thread et interface Runnable

- La classe Thread et son interface compagnon, Runnable sont emballés dans java.lang.
- Thread encapsule un thread d'exécution.
- Pour créer un nouveau thread: votre programme étendra Thread ou implémentera l'interface Runnable.

Method	Meaning
final String getName( )	Obtains a thread's name.
final int getPriority( )	Obtains a thread's priority.
final boolean isAlive( )	Determines whether a thread is still running.
final void join( )	Waits for a thread to terminate.
void run( )	Entry point for the thread.
static void sleep(long *milliseconds*)	Suspends a thread for a specified period of milliseconds.
void start( )	Starts a thread by calling its **run**( ) method.

Création d'un thread

- Un thread est créé en créant un objet de type `Thread`.
  - La classe `Thread` encapsule un objet exécutable.
- Deux façons de créer un objet exécutable:
  - Implémentez l'interface `Runnable`.
  - Étendez la classe `Thread`.
- N'oubliez pas:
  - Les deux approches utilisent toujours la classe `Thread` pour instancier, accéder et contrôler le thread. La seule différence est la façon dont une classe à thread activé est créée.

Interface Runnable

- L'interface `Runnable` abstrait une unité de code exécutable.
- Vous pouvez construire un thread sur n'importe quel objet qui implémente l'interface `Runnable`.
- `Runnable` définit une seule méthode:

```
public void run ()
```

  - Dans `run()`: vous définissez le code qui constitue le nouveau thread.
  - `run()` peut également appeler d'autres méthodes, utiliser d'autres classes et déclarer des variables comme le thread principal.
  - La seule différence: `run()` établit le point d'entrée pour un autre thread d'exécution simultané dans votre programme.
  - Ce thread se terminera lorsque `run()` retourne.

Classe Thread [1]

- Après avoir créé une classe qui implémente Runnable, vous créez un objet de type Thread sur un objet de cette classe.
- Constructeurs:

```
Thread()
Thread(String name)
Thread(Runnable threadOb)
Thread(Runnable threadOb, String name)
```

  - threadOb est une instance d'une classe qui implémente l'interface Runnable. Ceci définit où l'exécution du thread commencera.

Classe Thread [2]

- La méthode `start()`: `void start()` fait démarrer le nouveau thread quand il est appelé
- Essentiellement, `start()` exécute un appel à `run()`.
- La méthode `sleep()` provoque la suspension de l'exécution du thread à partir duquel il est appelé pendant la période spécifiée de millisecondes.

```
static void sleep(long milliseconds)
                    throws InterruptedException
```

- `millisecondes`: le nombre de millisecondes à suspendre.

```
1.  class MyThread implements Runnable {
2.      String thrdName;
3.      MyThread(String name) {
4.          thrdName = name;
5.      }
6.      // Entry point of thread.
7.      public void run() {
8.          System.out.println(thrdName + " starting.");
9.          try {
10.             for (int count = 0; count < 10; count++) {
11.                 Thread.sleep(400);
12.                 System.out.println("In " + thrdName +
13.                                    ", count is " + count);
14.             }
15.         } catch (InterruptedException exc) {
16.             System.out.println(thrdName + " interrupted.");
17.         }
18.         System.out.println(thrdName + " terminating.");
19.     }
20. }
```

```
1.  class UseThreads {
2.      public static void main(String args[]) {
3.          System.out.println("Main thread starting.");
4.          // First, construct a MyThread object.
5.          MyThread mt = new MyThread("Child #1");
6.          // Next, construct a thread from that object.
7.          Thread newThrd = new Thread(mt);
8.          // Finally, start execution of the thread.
9.          newThrd.start();
10.         for (int i = 0; i < 50; i++) {
11.             System.out.print(".");
12.             try {
13.                 Thread.sleep(100);
14.             } catch (InterruptedException exc) {
15.                 System.out.println("Main thread interrupted.");
16.             }
17.         }
18.         System.out.println("Main thread ending.");
19.     }
20. }
```

```
Main thread starting.
.Child #1 starting.
...In Child #1, count is 0
....In Child #1, count is 1
....In Child #1, count is 2
....In Child #1, count is 3
....In Child #1, count is 4
....In Child #1, count is 5
....In Child #1, count is 6
....In Child #1, count is 7
....In Child #1, count is 8
....In Child #1, count is 9
Child #1 terminating.
..........Main thread ending.
```

```
1.  class MyThread12 implements Runnable{
2.      Thread thrd;
3.      MyThread12(String name){ thrd = new Thread(this, name); }
4.      public static MyThread12 createAndStart(String name) {
5.          MyThread12 myThrd = new MyThread12(name);
6.          myThrd.thrd.start();
7.          return myThrd;
8.      }
9.      @Override
10.     public void run() {
11.         System.out.println(thrd.getName() + " starting.");
12.         try {
13.             for(int count = 0; count < 10; count++) {
14.                 Thread.sleep(400);
15.                 System.out.println("In " + thrd.getName() +
16.                                    ", count is " + count);
17.             }
18.         }catch(InterruptedException exc) {
19.           System.out.println(thrd.getName() + " interrupted.");
20.         }
21.         System.out.println(thrd.getName() + " terminating.");
22.     }
    }
```

```
1.  public class ThreadVariations {
2.      public static void main(String[] args) {
3.          System.out.println("Main thread starting.");
4.          // Create and start a thread.
5.          MyThread12 mt = MyThread12.createAndStart("Child #1");
6.          for(int i =0; i < 50; i++) {
7.              System.out.print(".");
8.              try {
9.                  Thread.sleep(100);
10.             }catch(InterruptedException exc) {
11.                 System.out.println("Main thread interruped.");
12.             }
13.         }
14.         System.out.println("Main thread ending.");
15.     }
16. }
```

```
1.  class MyThread1 extends Thread {
2.      MyThread1(String name) {
3.          super(name); // name thread
4.          start(); // start the thread
5.      }
6.      public void run() {
7.          System.out.println(getName() + " starting.");
8.          try {
9.              for (int count = 0; count < 10; count++) {
10.                 Thread.sleep(400);
11.                 System.out.println("In " + getName() +
12.                                    ", count is " + count);
13.             }
14.         }catch (InterruptedException exc) {
15.             System.out.println(getName() + " interrupted.");
16.         }
17.         System.out.println(getName() + " terminating.");
18.     }
19. }
```

```
1.  class ExtendThread {
2.      public static void main(String args[]) {
3.          System.out.println("Main thread starting.");
4.          MyThread1 mt = new MyThread1("Child #1");
5.          for (int i = 0; i < 50; i++) {
6.              System.out.print(".");
7.              try {
8.                  Thread.sleep(100);
9.              } catch (InterruptedException exc) {
10.                 System.out.println("Main thread interrupted.");
11.             }
12.         }
13.         System.out.println("Main thread ending.");
14.     }
15. }
```

Création de plusieurs threads

```
1.  class MyThread2 implements Runnable {
2.      Thread thrd;
3.      MyThread2(String name) {
4.          thrd = new Thread(this, name);
5.          thrd.start(); // start the thread
6.      }
7.      public void run() {
8.          System.out.println(thrd.getName() + " starting.");
9.          try {
10.             for (int count = 0; count < 10; count++) {
11.                 Thread.sleep(400);
12.                 System.out.println("In " + thrd.getName() +
13.                                    ", count is " + count);
14.             }
15.         } catch (InterruptedException exc) {
16.           System.out.println(thrd.getName() + " interrupted.");
17.         }
18.         System.out.println(thrd.getName() + " terminating.");
19.     }
20. }
```

```
1.  class MoreThreads {
2.      public static void main(String args[]) {
3.          System.out.println("Main thread starting.");
4.
5.          MyThread2 mt1 = new MyThread2("Child #1");
6.          MyThread2 mt2 = new MyThread2("Child #2");
7.          MyThread2 mt3 = new MyThread2("Child #3");
8.
9.          for (int i = 0; i < 50; i++) {
10.             System.out.print(".");
11.             try {
12.                 Thread.sleep(100);
13.             } catch (InterruptedException exc) {
14.                 System.out.println("Main thread interrupted.");
15.             }
16.         }
17.         System.out.println("Main thread ending.");
18.     }
19. }
```

```
Main thread starting.
Child #2 starting.
.Child #1 starting.
Child #3 starting.
...In Child #1, count is 0
In Child #3, count is 0
In Child #2, count is 0
....In Child #2, count is 1
In Child #1, count is 1
In Child #3, count is 1
....In Child #3, count is 2
In Child #2, count is 2
In Child #1, count is 2
....In Child #3, count is 3
In Child #2, count is 3
In Child #1, count is 3
....In Child #3, count is 4
In Child #2, count is 4
In Child #1, count is 4
....In Child #2, count is 5
In Child #3, count is 5
In Child #1, count is 5
....In Child #1, count is 6
In Child #3, count is 6
In Child #2, count is 6
....In Child #3, count is 7
In Child #2, count is 7
In Child #1, count is 7
....In Child #3, count is 8
In Child #2, count is 8
In Child #1, count is 8
....In Child #3, count is 9
Child #3 terminating.
In Child #2, count is 9
Child #2 terminating.
In Child #1, count is 9
Child #1 terminating.
..........Main thread ending.
```

Déterminer quand un thread se termine

- **Pour déterminer si un thread s'est terminé:**

```
final boolean isAlive()
```

  - Renvoie true si le thread sur lequel il est appelé est toujours en cours d'exécution.
  - Il renvoie faux sinon.

```
final void join() throws InterruptedException
```

  - Attend la fin du thread sur lequel il est appelé.
  - Des formes supplémentaires de `join()` vous permettent de spécifier une durée maximale pendant laquelle vous souhaitez attendre la fin du thread spécifié.

```
1.  class MoreThreads {
2.      /* version 2*/
3.      public static void main(String args[]) {
4.          System.out.println("Main thread starting.");
5.          MyThread2 mt1 = new MyThread2("Child #1");
6.          MyThread2 mt2 = new MyThread2("Child #2");
7.          MyThread2 mt3 = new MyThread2("Child #3");
8.          do {
9.            System.out.print(".");
10.           try { Thread.sleep(100);
11.           }catch(InterruptedException exc) {
12.             System.out.println("Main thread interrupted.");
13.           }
14.         } while (mt1.thrd.isAlive() ||
15.                  mt2.thrd.isAlive() ||
16.                  mt3.thrd.isAlive());
17.         System.out.println("Main thread ending.");
18.     }
19. }
```

```
1.  class MyThread3 implements Runnable {
2.      Thread thrd;
3.      MyThread3(String name) {
4.          thrd = new Thread(this, name);
5.          thrd.start(); // start the thread
6.      }
7.      public void run() {
8.          System.out.println(thrd.getName() + " starting.");
9.          try {
10.             for (int count = 0; count < 10; count++) {
11.                 Thread.sleep(400);
12.                 System.out.println("In " + thrd.getName() +
13.                                    ", count is " + count);
14.             }
15.         } catch (InterruptedException exc) {
16.          System.out.println(thrd.getName() + " interrupted.");
17.         }
18.         System.out.println(thrd.getName() + " terminating.");
19.     }
20. }
```

```
1.  class JoinThreads {
2.      public static void main(String args[]) {
3.          System.out.println("Main thread starting.");
4.          MyThread3 mt1 = new MyThread3("Child #1");
5.          MyThread3 mt2 = new MyThread3("Child #2");
6.          MyThread3 mt3 = new MyThread3("Child #3");
7.          try {
8.              mt1.thrd.join();
9.              System.out.println("Child #1 joined.");
10.             mt2.thrd.join();
11.             System.out.println("Child #2 joined.");
12.             mt3.thrd.join();
13.             System.out.println("Child #3 joined.");
14.         } catch (InterruptedException exc) {
15.             System.out.println("Main thread interrupted.");
16.         }
17.         System.out.println("Main thread ending.");
18.     }
19. }
```

```
Main thread starting.
Child #1 starting.
Child #2 starting.
Child #3 starting.
In Child #2, count is 0
In Child #1, count is 0
In Child #3, count is 0
In Child #3, count is 1
In Child #1, count is 1
In Child #2, count is 1
In Child #3, count is 2
In Child #2, count is 2
In Child #1, count is 2
In Child #1, count is 3
In Child #3, count is 3
In Child #2, count is 3
In Child #1, count is 4
In Child #2, count is 4
In Child #3, count is 4
In Child #1, count is 5
In Child #2, count is 5
In Child #3, count is 5
In Child #1, count is 6
In Child #3, count is 6
In Child #2, count is 6
In Child #3, count is 7
In Child #1, count is 7
In Child #2, count is 7
In Child #2, count is 8
In Child #3, count is 8
In Child #1, count is 8
In Child #3, count is 9
In Child #2, count is 9
Child #2 terminating.
Child #3 terminating.
In Child #1, count is 9
Child #1 terminating.
Child #1 joined.
Child #2 joined.
Child #3 joined.
Main thread ending.
```

Priorités des threads [1]

- Chaque thread a associé un paramètre de priorité.
  - La priorité d'un thread détermine, en partie, le temps CPU reçu par un thread par rapport aux autres threads actifs.
- En général: sur une période de temps donnée, les threads de faible priorité reçoivent peu. Les threads hautement prioritaires en reçoivent beaucoup.
- Lorsqu'un thread enfant est démarré, son paramètre de priorité est égal à celui de son thread parent.
- Vous pouvez modifier la priorité d'un thread

```
final void setPriority(int level)
```

- `level` spécifie le nouveau paramètre de priorité pour le thread appelant.

Priorités des threads [2]

- La valeur du niveau doit être comprise dans la plage `MIN_PRIORITY` (1) et `MAX_PRIORITY` (10).
- Priorité par défaut: `NORM_PRIORITY` (5)

- Pour obtenir le réglage de priorité actuel:

```
final int getPriority()
```

```
1.  class Priority implements Runnable {
2.      int count; Thread thrd;
3.      static boolean stop = false; static String currentName;
4.      Priority(String name) {
5.          thrd = new Thread(this, name); count = 0;
6.          currentName = name;
7.      }
8.      public void run() {
9.          System.out.println(thrd.getName() + " starting.");
10.         do {
11.             count++;
12.             if (currentName.compareTo(thrd.getName()) != 0) {
13.                 currentName = thrd.getName();
14.                 System.out.println("In " + currentName);
15.             }
16.         } while (stop == false && count < 10000000);
17.         stop = true;
18.       System.out.println("\n"+thrd.getName()+" terminating.");
19.     }
20. }
```

```
1.  class PriorityDemo {
2.     public static void main(String args[]) {
3.        Priority mt1 = new Priority("High Priority");
4.        Priority mt2 = new Priority("Low Priority");
5.        Priority mt3 = new Priority("Normal Priority #1");
6.        Priority mt4 = new Priority("Normal Priority #2");
7.        Priority mt5 = new Priority("Normal Priority #3");
8.        mt1.thrd.setPriority(Thread.NORM_PRIORITY + 2);
9.        mt2.thrd.setPriority(Thread.NORM_PRIORITY - 2);
10.       mt1.thrd.start(); mt2.thrd.start();
11.       mt3.thrd.start(); mt4.thrd.start(); mt5.thrd.start();
12.       try {
13.          mt1.thrd.join(); mt2.thrd.join();
14.          mt3.thrd.join(); mt4.thrd.join(); mt5.thrd.join();
15.       } catch (InterruptedException exc) {
16.          System.out.println("Main thread interrupted.");
17.       }
18.       // ....
```

```
1.          // ...
2.          System.out.println("\nHigh priority thread counted to "
3.                                                  + mt1.count);
4.          System.out.println("Low priority thread counted to " +
5.                                                  mt2.count);
6.          System.out.println("1st Normal priority thread" +
7.                                          " counted to " + mt3.count);
8.          System.out.println("2nd Normal priority thread " +
9.                                          " counted to " + mt4.count);
10.         System.out.println("3rd Normal priority thread " +
11.                                         " counted to " + mt5.count);
12.     }
13. }
```

```
High priority thread counted to 10000000
Low priority thread counted to 6444752
1st Normal priority thread counted to 7071170
2nd Normal priority thread counted to 9116802
3rd Normal priority thread counted to 7330314
```

Plan du cours

1. Fondamentaux du multithreading
2. Synchronisation

Synchronization [1]

- Il est parfois nécessaire de coordonner les activités de deux threads ou plus → synchronisations.
- Raison de l'utilisation de la synchronisation:
  - Lorsque deux threads ou plus ont besoin d'accéder à une ressource partagée qui peut être utilisée par un seul thread à la fois.
  - Lorsqu'un thread attend un événement provoqué par un autre thread.
- Clé de synchronisation: moniteur (contrôle l'accès à un objet)
  - Un moniteur fonctionne en mettant en œuvre le concept d'une serrure (lock).
  - Lorsqu'un objet est verrouillé par un thread, aucun autre thread ne peut accéder à l'objet. Lorsque le thread se termine, l'objet est déverrouillé et est disponible pour être utilisé par un autre thread.

Synchronization [2]

- Tous les objets en Java ont un moniteur.
  - Cette fonctionnalité est intégrée au langage Java lui-même.

  → tous les objets peuvent être synchronisés.
- Mot-clé: `synchronized`.
- Deux façons de synchroniser:
  - méthode synchronisée et instruction synchronisée.
  - les deux utilisent le mot-clé `synchronized`.

Méthodes synchronisées

- Utilisez le mot-clé `synchronized`.
- **Lorsque cette méthode est appelée**: le thread appelant entre dans le moniteur de l'objet, qui verrouille ensuite l'objet.
- **Lorsqu'il est verrouillé**: aucun autre thread ne peut entrer dans la méthode, ni entrer aucune autre méthode synchronisée définie par la classe de l'objet.
- **Lorsque le thread retourne de la méthode**: le moniteur déverrouille l'objet, lui permettant d'être utilisé par le thread suivant.

→ la synchronisation est réalisée sans pratiquement aucun effort de programmation de votre part.

```
1.  class SumArray {
2.      private int sum;
3.      synchronized int sumArray(int nums[]) {
4.          sum = 0; // reset sum
5.          for (int i = 0; i < nums.length; i++) {
6.              sum += nums[i];
7.              System.out.println("Running total for " +
8.                Thread.currentThread().getName() + " is " + sum);
9.              try {
10.                 Thread.sleep(10); // allow task-switch
11.             } catch (InterruptedException exc) {
12.                 System.out.println("Thread interrupted.");
13.             }
14.         }
15.         return sum;
16.     }
17. }
```

```
1.  class MyThread4 implements Runnable {
2.      Thread thrd;
3.      static SumArray sa = new SumArray();
4.      int a[]; int answer;
5.
6.      MyThread4(String name, int nums[]) {
7.          thrd = new Thread(this, name);
8.          a = nums; thrd.start(); // start the thread
9.      }
10.     public void run() {
11.         int sum;
12.         System.out.println(thrd.getName() + " starting.");
13.
14.         answer = sa.sumArray(a);
15.         System.out.println("Sum for " + thrd.getName() +
16.                                          " is " + answer);
17.         System.out.println(thrd.getName() + " terminating.");
18.     }
19. }
```

```
1.  class Sync {
2.      public static void main(String args[]) {
3.          int a[] = { 1, 2, 3, 4, 5 };
4.
5.          MyThread4 mt1 = new MyThread4("Child #1", a);
6.          MyThread4 mt2 = new MyThread4("Child #2", a);
7.
8.          try {
9.              mt1.thrd.join();
10.             mt2.thrd.join();
11.         } catch (InterruptedException exc) {
12.             System.out.println("Main thread interrupted.");
13.         }
14.     }
15. }
```

```
Child #2 starting.
Running total for Child #2 is 1
Child #1 starting.
Running total for Child #2 is 3
Running total for Child #2 is 6
Running total for Child #2 is 10
Running total for Child #2 is 15
Running total for Child #1 is 1
Sum for Child #2 is 15
Child #2 terminating.
Running total for Child #1 is 3
Running total for Child #1 is 6
Running total for Child #1 is 10
Running total for Child #1 is 15
Sum for Child #1 is 15
Child #1 terminating.
```

Instruction synchronisée

- Bien que la création de méthodes synchronisées dans les classes que vous créez soit un moyen simple et efficace de réaliser la synchronisation, cela ne fonctionnera pas dans tous les cas.
- Exemple: vous souhaiterez peut-être synchroniser l'accès à une méthode qui n'est pas modifiée par `synchronized`.
  - Cela peut se produire car vous souhaitez utiliser une classe qui n'a pas été créée par vous mais par un tiers et vous n'avez pas accès au code source.

→ Il n'est pas possible pour vous d'ajouter synchronisé aux méthodes appropriées au sein de la classe: il suffit de placer des appels aux méthodes définies par cette classe dans un bloc synchronisé.

- Forme générale d'un bloc synchronisé :

```
synchronized(objref) {
   // statements to be synchronized
}
```

  - `objref` est une référence à l'objet en cours de synchronisation.
  - Une fois qu'un bloc synchronisé a été entré, aucun autre thread ne peut appeler une méthode synchronisée sur l'objet référencé par `objref` tant que le bloc n'a pas été quitté.

```
1.  class SumArray1 {
2.      private int sum;
3.      int sumArray1(int nums[]) {
4.          sum = 0; // reset sum
5.          for (int i = 0; i < nums.length; i++) {
6.              sum += nums[i];
7.              System.out.println("Running total for " +
8.              Thread.currentThread().getName() + " is " + sum);
9.              try {
10.                 Thread.sleep(10); // allow task-switch
11.             } catch (InterruptedException exc) {
12.                 System.out.println("Thread interrupted.");
13.             }
14.         }
15.         return sum;
16.     }
17. }
```

```
1.  class MyThread5 implements Runnable {
2.      Thread thrd;
3.      static SumArray sa = new SumArray();
4.      int a[]; int answer;
5.      MyThread5(String name, int nums[]) {
6.          thrd = new Thread(this, name);
7.          a = nums;
8.          thrd.start(); // start the thread
9.      }
10.     public void run() {
11.         int sum;
12.         System.out.println(thrd.getName() + " starting.");
13.         // synchronize calls to sumArray()
14.         synchronized (sa) { answer = sa.sumArray(a); }
15.         System.out.println("Sum for " + thrd.getName() +
16.                                          " is " + answer);
17.         System.out.println(thrd.getName() + " terminating.");
18.     }
19. }
```

```
1.  class Sync1 {
2.      public static void main(String args[]) {
3.          int a[] = { 1, 2, 3, 4, 5 };
4.          MyThread5 mt1 = new MyThread5("Child #1", a);
5.          MyThread5 mt2 = new MyThread5("Child #2", a);
6.
7.          try {
8.              mt1.thrd.join();
9.              mt2.thrd.join();
10.         } catch (InterruptedException exc) {
11.             System.out.println("Main thread interrupted.");
12.         }
13.     }
14. }
```

Communication entre les threads

- Un thread appelé **T** s'exécute à l'intérieur d'une méthode synchronisée et a besoin d'accéder à une ressource appelée **R** qui est temporairement indisponible. Que dois-je faire?
  - Si **T** entre dans une forme de boucle d'interrogation qui attend **R**, **T** attache l'objet, empêchant les autres threads d'y accéder → annule partiellement les avantages de la programmation pour un environnement multithread.
  - Une meilleure solution: laisser T abandonner temporairement le contrôle de l'objet, permettant à un autre thread de s'exécuter. Lorsque **R** devient disponible, **T** peut être notifié et reprendre l'exécution → communication entre les threads (un thread peut notifier à un autre qu'il est bloqué et être averti qu'il peut reprendre l'exécution).
- Java prend en charge la communication entre les threads avec les méthodes `wait()`, `notify()` et `notifyAll()`.

```
final void wait()
          throws InterruptedException
final void wait(long millis)
          throws InterruptedException
final void wait(long millis, int nanos)
          throws InterruptedException
final void notify()
```

reprend un thread en attente

```
final void notifyAll()
```

notifie tous les threads, le planificateur détermine quel thread accède à l'objet.

```
1.  class TickTock {
2.      String state; // contains the state of the clock
3.      synchronized void tick(boolean running) {
4.          if (!running) { state = "ticked"; notify(); return; }
5.          System.out.print("Tick ");
6.          state = "ticked";  notify();
7.          try { while (!state.equals("tocked")) wait();
8.          } catch (InterruptedException exc) {
9.              System.out.println("Thread interrupted.");
10.         }
11.     }
12.     synchronized void tock(boolean running) {
13.         if (!running) { state = "tocked"; notify();  return; }
14.         System.out.println("Tock");
15.         state = "tocked";  notify();
16.         try { while (!state.equals("ticked")) wait();
17.         } catch (InterruptedException exc) {
18.             System.out.println("Thread interrupted.");
19.         }
20.     }
21. }
```

```
1.  class MyThread6 implements Runnable {
2.      Thread thrd;
3.      TickTock ttOb;
4.      MyThread6(String name, TickTock tt) {
5.          thrd = new Thread(this, name);
6.          ttOb = tt;
7.          thrd.start(); // start the thread
8.      }
9.
10.     public void run() {
11.         if (thrd.getName().compareTo("Tick") == 0) {
12.             for (int i = 0; i < 5; i++)
13.                 ttOb.tick(true);
14.             ttOb.tick(false);
15.         } else {
16.             for (int i = 0; i < 5; i++)
17.                 ttOb.tock(true);
18.             ttOb.tock(false);
19.         }
20.     }
21. }
```

```
1.  class ThreadCom {
2.      public static void main(String args[]) {
3.          TickTock tt = new TickTock();
4.          MyThread6 mt1 = new MyThread6("Tick", tt);
5.          MyThread6 mt2 = new MyThread6("Tock", tt);
6.
7.
8.          try {
9.              mt1.thrd.join();
10.             mt2.thrd.join();
11.         } catch (InterruptedException exc) {
12.             System.out.println("Main thread interrupted.");
13.         }
14.     }
15. }
```

Deadlock, race condition

- Deadlock est une situation dans laquelle un thread attend qu'un autre thread fasse quelque chose, mais cet autre thread attend le premier → les deux threads sont suspendus, s'attendent l'un l'autre et aucun ne s'exécute.
- Une race condition se produit lorsque deux threads (ou plus) tentent d'accéder à une ressource partagée en même temps, sans synchronisation appropriée.

Suspension, reprise et arrêt des threads

- `final void resume()`
- `final void suspend()`
- `final void stop()`
- Bien que ces méthodes semblent être une approche parfaitement raisonnable et pratique pour gérer l'exécution des threads, elles ne doivent plus être utilisées.
  - Déconseillé par Java 2.

```
1.  class MyThread8 implements Runnable {
2.      Thread thrd; boolean suspended; boolean stopped;
3.      MyThread8(String name) {
4.          thrd = new Thread(this, name); suspended = false;
5.          stopped = false; thrd.start();
6.      }
7.      public void run() {
8.          System.out.println(thrd.getName() + " starting.");
9.          try {
10.             for (int i = 1; i < 1000; i++) {
11.                 System.out.print(i + " ");
12.                 if ((i % 10) == 0) { System.out.println();
13.                     Thread.sleep(250);
14.                 }
15.                 synchronized (this) {while(suspended){wait();}
16.                     if (stopped) break;
17.                 }
18.             }
19.         } catch (InterruptedException exc) {
20.             System.out.println(thrd.getName()+" interrupted.");
21.         }
```

```
1.          System.out.println(thrd.getName() + " exiting.");
2.      }
3.      synchronized void mystop() {
4.          stopped = true;
5.          suspended = false;
6.          notify();
7.      }
8.      synchronized void mysuspend() {
9.          suspended = true;
10.     }
11.     synchronized void myresume() {
12.         suspended = false;
13.         notify();
14.     }
15. }
```

```
1.  class Suspend {
2.     public static void main(String args[]) {
3.        MyThread8 ob1 = new MyThread8("My Thread");
4.        try {
5.           Thread.sleep(1000);
6.           // let ob1 thread start executing
7.
8.           ob1.mysuspend();
9.           System.out.println("Suspending thread.");
10.          Thread.sleep(1000);
11.
12.          ob1.myresume();
13.          System.out.println("Resuming thread.");
14.          Thread.sleep(1000);
15.
16.          ob1.mysuspend();
17.          System.out.println("Suspending thread.");
18.          Thread.sleep(1000);
```

```
1.              ob1.myresume();
2.              System.out.println("Resuming thread.");
3.              Thread.sleep(1000);
4.
5.              ob1.mysuspend();
6.              System.out.println("Stopping thread.");
7.              ob1.mystop();
8.          } catch (InterruptedException e) {
9.              System.out.println("Main thread Interrupted");
10.         }
11.         try {
12.             ob1.thrd.join();
13.         } catch (InterruptedException e) {
14.             System.out.println("Main thread Interrupted");
15.         }
16.
17.         System.out.println("Main thread exiting.");
18.     }
19. }
```

```
My Thread starting.
1 2 3 4 5 6 7 8 9 10
11 12 13 14 15 16 17 18 19 20
21 22 23 24 25 26 27 28 29 30
31 32 33 34 35 36 37 38 39 40
Suspending thread.
Resuming thread.
41 42 43 44 45 46 47 48 49 50
51 52 53 54 55 56 57 58 59 60
61 62 63 64 65 66 67 68 69 70
71 72 73 74 75 76 77 78 79 80
Suspending thread.
Resuming thread.
81 82 83 84 85 86 87 88 89 90
91 92 93 94 95 96 97 98 99 100
101 102 103 104 105 106 107 108 109 110
111 112 113 114 115 116 117 118 119 120
Stopping thread.
My Thread exiting.
Main thread exiting.
```

Exercices

- **Exo 1**: Écrivez un programme qui renvoie le nombre de diviseurs pour un entier entré par l'utilisateur.
- **Exo 2:** (Réutilisez **Exo1**) Écrivez un programme qui renvoie l'entier ayant le plus grand nombre de diviseurs pour une intervalle donnée [1, 10000]. Affichez cet entier et son nombre de diviseurs.
- **Exo 3**: (Réutilisez **Exo 2**) Écrivez un programme qui renvoie l'entier ayant le plus grand nombre de diviseurs pour une intervalle donnée [1, 100000]. Affichez cet entier et son nombre de diviseurs. **Remarque**: Dans cet exercice, demandez à l'utilisateur d'entrer le nombre de threads. Le programme divisera le problème en plusieurs parties et créera un thread pour résoudre chaque partie du problème. (Utilisez la programmation multithread).

QUESTION ?