# TD2 : Java « avancé » (correction)
# V1.3.0



Document en ligne : www.mickael-martin-nevot.com

## 1 Généralités

Écrivez les applications ci-dessous en Java et en respectant la norme de programmation donnée en cours puis testez-les.

## 2 Création de threads

En Java, il existe deux façons de créer des *threads*. La première, en étendant la classe `Thread` :

```
public class MyThread extends Thread {
    public void run() {
        // Ici se trouve la description du comportement du thread.
    }
    public static void main(String[] args) {
        // On crée une instance de l'objet MyThread.
        MyThread t1 = new MyThread();
        // Puis on lance le thread : t1 exécute alors la méthode run().
        t1.start();
    }
}
```

La seconde façon, en implémentant l'interface `Runnable` :

```
public class MyJob implements Runnable {
    public void run() {
        // Ici se trouve la description du comportement du thread.
    }
    public static void main(String[] args) {
        // On crée une tâche pour un thread.
        MyJob job = new MyJob();
        // On crée une instance de Thread avec la tâche job.
        Thread t1 = new Thread(job);
        // Puis on lance le thread : t1 exécute alors la méthode run()
        // de la tâche job.
        t1.start();
    }
}
```

Déterminez pourquoi il existe deux manières de faire et quelles sont les différences entre elles.

*Correction*

Une classe ne peut étendre qu'une seule autre classe. La première méthode est donc très contraignante. De plus, d'un point de vue conception, il est plus propre de définir des tâches, puis des threads exécutant ces tâches. La deuxième méthode est donc celle à utiliser.

## 3 Threads et JVM

Écrivez en Java un programme qui utilise deux *threads* en parallèle :

- le premier affichera les 26 lettres de l'alphabet (dans l'ordre lexicographique) ;
- le second affichera les nombres de 1 à 26 (dans l'ordre lexicographique).

Déterminez le résultat de ce programme lors de son exécution.

*Correction*

```
public class Thread1 implements Runnable {
    public void run() {
        try {
            for(int i = 1 ; i <= 26 ; ++i) {
                System.out.println(i + " ");
            }
        } catch (InterruptedException e) {
            return;
        }
    }
}

public class Thread2 implements Runnable {
    public void run() {
        try {
            for(int i = 'a' ; i <= 'z' ; ++i) {
                System.out.println(i + " ");
            }
        } catch (InterruptedException e) {
            return;
        }
    }
}

public class MyApplication {
    public static void main(String[] args) {
        Runnable t1 = new Thread1();
        Runnable t2 = new Thread2();
        new Thread(t1).start();
        new Thread(t2).start();
    }
}
```

L'ordonnanceur du système d'exploitation gère le processus de la JVM. L'ordonnanceur de la JVM gère les threads prêts à s'exécuter (ainsi que le ramasse miettes). Il n'y a aucun moyen pour l'utilisateur d'agir sur l'ordonnanceur de la JVM.

## 4 Méthodes de la classe Thread

Écrivez un programme dans lequel un *thread* principal lance trois nouveaux *threads* qui effectuent dix fois les actions suivantes (dans l'ordre) :

- attendre un temps aléatoire compris entre 0 ms et 200 ms ;
- afficher son nom.

Le *thread* principal doit attendre la fin de l'exécution des trois *threads* avant de terminer son exécution.

*Correction*

```
// Méthode 1 : Thread.
Public class MyThread extends Thread {
    public void run() {
        for (int i = 0; i < 10; ++i) {
            // Le thread en cours d'exécution attend entre 0 et 200 ms.
            try {
                Thread.currentThread().sleep((int) Math.random () * 200);
            } catch(InterruptedException e) {
                return;
            }
            // Puis il écrit son nom.
            System.out.println("Je suis " + Thread.currentThread().getName());
        }
    }
}

// Méthode 2 : Runnable.
public class Job implements Runnable {
    public void run() {
        for (int i = 0; i < 10; ++i) {
            // Le thread en cours d'exécution attend entre 0 et 200 ms.
            try {
                Thread.currentThread().sleep((int) Math.random () * 200);
            } catch (InterruptedException e) {
                return;
            }
            // Puis il écrit son nom.
            System.out.println("Je suis " + Thread.currentThread().getName());
        }
    }
}

public class TestThread {
    public static void main(String [] args) {
        // Déclare 3 threads.
        Thread t1;
        Thread t2;
        Thread t3;

        // Méthode 1 : en héritant de la class Thread.
        System.out.println("Avec la méthode 1");
        // Crée 3 threads suivant le modèle défini dans MyThread.
        // Leur comportement est celui du run de la class MyThread.
```

```
        t1 = new MyThread();
        t2 = new MyThread();
        t3 = new MyThread();
        // Donne un nom à chaque thread.
        t1.setName("Tom");
        t2.setName("Pierre");
        t3.setName("Lucie");
        // Lance les threads.
        t1.start();
        t2.start();
        t3.start();
        try {
            // Le thread main attend la fin d'exécution de t1, t2 et t3.
            t1.join();
            t2.join();
            t3.join();
        } catch (InterruptedException e) {
            return;
        }
        System.out.println();
        System.out.println();

        // Méthode 2 : en implémentant l'interface Runnable.
        System.out.println("Avec la méthode 2");
        // Crée une tâche à exécuter pour les threads.
        Job job = new Job ();
        // Crée 3 threads avec le comportement défini par l'objet job.
        // Leur comportement est donc celui du run de la classe Job.
        t1 = new Thread(job);
        t2 = new Thread(job);
        t3 = new Thread(job);
        // Donne un nom à chaque thread.
        t1.setName("Jerry");
        t2.setName("Luc");
        t3.setName("Namie");
        // Lance les threads.
        t1.start();
        t2.start();
        t3.start();
        try {
            // Le thread main attend la fin d'exécution de t1, t2 et t3.
            t1.join();
            t2.join();
            t3.join();
        } catch (InterruptedException e) {
            return ;
        }
    }
}
```

## 5 Threads et concurrence

Vous allez mettre en évidence ce qui peut se passer quand deux *threads* (représentés par deux personnes : Juliette et Roméo) partagent un même objet (représenté par un compte en banque).

## 5.1 Classe Account

La classe `Account` contient :

- la variable d'instance : `balance` (initialisée à 100, représentant le solde courant du compte) ;
- la méthode `withdraw(int amount)` permettant de faire un retrait.

## 5.2 Classe JulietteAndRomeoJob

La classe `JulietteAndRomeoJob` est une tâche (elle implémente l'interface `Runnable`) et représente aussi bien Juliette que Roméo. Elle contient :

- les variables d'instance : `name`, `account` (de type `Account`) ;
- la méthode `doWithdraw(int amount)` permettant à Roméo ou à Juliette d'effectuer un retrait sur leur compte en banque : la personne souhaitant le faire vérifie `account` (et affiche sa valeur), s'endort durant 500 ms, puis (à son réveil) effectue le retrait en signalant son nom ;
- la méthode `run()` permettant d'effectuer dix retraits de 10 € ;
- la méthode statique `main(String[] args)` permettant de créer deux tâches `romeoJob` et `julietteJob` qui seront utilisées par deux *threads* `romeo` et `juliette` (de type `JulietteAndRomeoJob`), de les nommer puis de les exécuter.

## 5.3 Test

Après avoir écrit le programme, testez-le et examinez son comportement.

*Correction*

```
public class Account {
    private int balance = 100;

    public int getBalance () {
        return this.balance;
    }

    public void withdraw(int amount){
        this.balance -= amount;
    }
}

public class JulietteAndRomeoJob implements Runnable {
    private String name;
    private Account account;

    public JulietteAndRomeoJob() {
        this.account = new Account();
    }

    private void doWithdraw(int amount) {
        String name = Thread.currentThread().getName();
        System.out.println(this.name + " va retirer");
        try {
            System.out.println(this.name + " va dormir");
            Thread.sleep(500);
        } catch (InterruptedException e) {}
```

```
        System.out.println(this.name + " se réveille");
        account.withdraw(this.amount);
        System.out.println(this.name + " a fini de retirer");
    }

    public void run() {
        for (int x = 0; x < 10; ++x) {
            doWithdraw(10);
            if (compte.getBalance() < 0) {
                System.out.println("découvert !");
            }
        }
    }

    public static void main(String [] args) {
        JulietteAndRomeoJob job = new JulietteAndRomeoJob();
        Thread t1 = new Thread(job);
        Thread t2 = new Thread(job);
        t1.setName("Romeo");
        t2.setName("Juliette");
        t1.start();
        t2.start();
    }
}
```