Chapitre X

threads

Threads

- threads: plusieurs activités qui coexistent et partagent des données
  - exemples:
    - pendant un chargement long faire autre chose, coopérer
  - processus versus threads: partage de mémoire
  - problème de l'accès aux ressources partagées
    - verrous
    - moniteurs
    - synchronisation

Principes de base

- extension de la classe Thread
  - méthode `run` est le code qui sera exécuté.
  - la création d'un objet dont la superclasse est `Thread` crée la thread (mais ne la démarre pas)
  - la méthode `start` démarre la thread (et retourne immédiatement)
  - la méthode `join` permet d'attendre la fin de la thread
  - les exécutions des threads sont asynchrones et concurrentes

Exemple

```
class ThreadAffiche extends Thread{
    private String mot;
    private int delay;
    public ThreadAffiche(String w,int duree){
        mot=w;
        delay=duree;
    }
    public void run(){
        try{
            for(;;){
                System.out.println(mot);
                Thread.sleep(delay);
            }
        }catch(InterruptedException e){
        }
    }
}
public static void main(String[] args) {
       new ThreadAffiche("PING", 10).start();
       new ThreadAffiche("PONG", 30).start();
       new ThreadAffiche("Splash!",60).start();
  }
```

Alternative: Runnable

- Une autre solution:
  - créer une classe qui implémente l'interface `Runnable` (cette interface contient la méthode `run`)
  - créer une `Thread` à partir du constructeur `Thread` avec un `Runnable` comme argument.

Exemple

```
class RunnableAffiche implements Runnable{
    private String mot;
    private int delay;
    public RunnableAffiche(String w,int duree){
        mot=w;
        delay=duree;
    }
    public void run(){
        try{
            for(;;){
                System.out.println(mot);
                Thread.sleep(delay);
            }
        }catch(InterruptedException e){
        }
    }
}
public static void main(String[] args) {
     Runnable ping=new RunnableAffiche("PING", 10);
     Runnable pong=new RunnableAffiche("PONG", 50);
     new Thread(ping).start();
     new Thread(pong).start();
}
```

Synchronisation

- les threads s'exécutent concurremment et peuvent accéder concurremment à des objets:
  - il faut contrôler l'accès:
    - thread un lit une variable (R1) puis modifie cette variable (W1)
    - thread deux lit la même variable (R2) puis la modifie (W2)
    - R1-R2-W2-W1
    - R1-W1-R2-W2 résultat différent!

Exemple

```
class X{
    int val;
}
class Concur extends Thread{
    X x;
    int i;
    String nom;
    public Concur(String st, X x){
        nom=st;
        this.x=x;
    }
    public void run(){
        i=x.val;
        System.out.println("thread:"+nom+" valeur x="+i);
        try{
        Thread.sleep(10);
        }catch(Exception e){}
        x.val=i+1;
        System.out.println("thread:"+nom+" valeur x="+x.val);
    }
}
```

Suite

```
public static void main(String[] args) {
        X x=new X();
        Thread un=new Concur("un",x);
        Thread deux=new Concur("deux",x);
        un.start(); deux.start();
        try{
            un.join();
            deux.join();
        }catch (InterruptedException e){}
        System.out.println("X="+x.val);
 }
```

donnera (par exemple)

- thread:un valeur x=0
- thread:deux valeur x=0
- thread:un valeur x=1
- thread:deux valeur x=1
- X=1

Deuxième exemple

```
class Y{
    int val=0;
    public int increment(){
        int tmp=val;
        tmp++;
        try{
            Thread.currentThread().sleep(100);
        }catch(Exception e){}
        val=tmp;
        return(tmp);
    }
    int getVal(){return val;}
}
class Concur1 extends Thread{
    Y y;
    String nom;
    public Concur1(String st, Y y){
        nom=st;
        this.y=y;
    }
    public void run(){
        System.out.println("thread:"+nom+" valeur="+y.increment());
    }
}
```

Suite

```
  public static void main(String[] args) {
        Y y=new Y();
        Thread un=new Concur1("un",y);
        Thread deux=new Concur1("deux",y);
        un.start(); deux.start();
        try{
            un.join();
            deux.join();
        }catch (InterruptedException e){}
        System.out.println("Y="+y.getVal());
    }
----------
```

- thread:un valeur=1
- thread:deux valeur=1
- Y=1

Verrous

- à chaque objet est associé un verrou
  - synchronized(expr) {instructions}
    - expr doit s'évaluer comme une référence à un objet
    - verrou sur cet objet pour la durée de l'exécution de instructions
  - déclarer les méthodes comme synchronized: la thread obtient le verrou et le relâche quand la méthode se termine

synchronised(x)

```
class Concur extends Thread{
    X x;
    int i;
    String nom;
    public Concur(String st, X x){
        nom=st;
        this.x=x;
    }
    public void run(){
        synchronized(x){
            i=x.val;
            System.out.println("thread:"+nom+" valeur x="+i);
            try{
                Thread.sleep(10);
            }catch(Exception e){}
            x.val=i+1;
            System.out.println("thread:"+nom+" valeur x="+x.val);
        }
    }
}
```

Méthode synchronisée

```
class Y{
    int val=0;
    public synchronized int increment(){
        int tmp=val;
        tmp++;
        try{
            Thread.currentThread().sleep(100);
        }catch(Exception e){}
        val=tmp;
        return(tmp);
    }
    int getVal(){return val;}
}
------------
```

- thread:un valeur=1
- thread:deux valeur=2
- Y=2

Mais…

- la synchronisation par des verrous peut entraîner un blocage:
  - la thread un (XA) pose un verrou sur l'objet A et (YB) demande un verrou sur l'objet B
  - la thread deux (XB) pose un verrou sur l'objet B et (YA) demande un verrou sur l'objet A
  - si XA –XB : ni YA ni YB ne peuvent êter satisfaites -> blocage
- (pour une méthode synchronisée, le verrou concerne l'objet globalement et pas seulement la méthode)

Exemple

```
class Dead{
    Dead partenaire;
    String nom;
    public Dead(String st){
        nom=st;
    }
    public synchronized void f(){
        try{
            Thread.currentThread().sleep(100);
        }catch(Exception e){}
        System.out.println(Thread.currentThread().getName()+
                " de "+ nom+".f() invoque "+ partenaire.nom+".g()");
        partenaire.g();    }
    public synchronized void g(){
        System.out.println(Thread.currentThread().getName()+
                " de "+ nom+".g()");
    }
    public void setPartenaire(Dead d){
        partenaire=d;
    }
}
```

Exemple (suite)

```
        final Dead un=new Dead("un");
        final Dead deux= new Dead("deux");
        un.setPartenaire(deux);
        deux.setPartenaire(un);
        new Thread(new Runnable(){public void run(){un.f();}
        },"T1").start();
        new Thread(new Runnable(){public void run(){deux.f();}
        },"T2").start();
------------
```

- T1 de un.f() invoque deux.g()
- T2 de deux.f() invoque un.g()

Synchronisation…

- wait, notifyAll notify
  - attendre une condition / notifier le changement de condition:

```
synchronized void fairesurcondition(){
    while(!condition){
        wait();
        faire ce qu'il faut quand la condition est vraie
        }
-----------------
synchronized void changercondition(){
    … changer quelque chose concernant la condition
    notifyAll(); // ou notify()
}
```

Exemple (file: rappel Cellule)

```
public class Cellule<E>{
    private Cellule<E> suivant;
    private E element;
    public Cellule(E val) {
        this.element=val;
    }
    public Cellule(E val, Cellule suivant){
        this.element=val;
        this.suivant=suivant;
    }
    public E getElement(){
        return element;
    }
    public void setElement(E v){
        element=v;
    }
    public Cellule<E> getSuivant(){
        return suivant;
    }
    public void setSuivant(Cellule<E> s){
        this.suivant=s;
    }
```

File synchronisées

```
class File<E>{
    protected Cellule<E> tete, queue;
    private int taille=0;

    public synchronized void enfiler(E item){
        Cellule<E> c=new Cellule<E>(item);
        if (queue==null)
            tete=c;
        else{
            queue.setSuivant(c);
        }
        c.setSuivant(null);
        queue = c;
        notifyAll();
 }
```

File (suite)

```
public synchronized E defiler() throws InterruptedException{
       while (tete == null)
           wait();
       Cellule<E> tmp=tete;
       tete=tete.getSuivant();
       if (tete == null) queue=null;
       return tmp.getElement();
}
```