

Cours du 3/12

Threads



Threads

- threads: plusieurs activités qui coexistent et partagent des données
 - exemples:
 - pendant un chargement long faire autre chose
 - coopérer
 - processus versus threads
 - problème de l'accès aux ressources partagées
 - verrous
 - moniteur
 - 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){
        }
    }
}
```



Suite

```
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){
        }
    }
}
```




Suite

```
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
concurrentement et peuvent accéder
concurrentement 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());
    }
}
```

}thread

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
 - Pose un 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);
        }
    }
}
thread
```


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 être 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;
    }
}
```

Thread

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();
    }
}
```



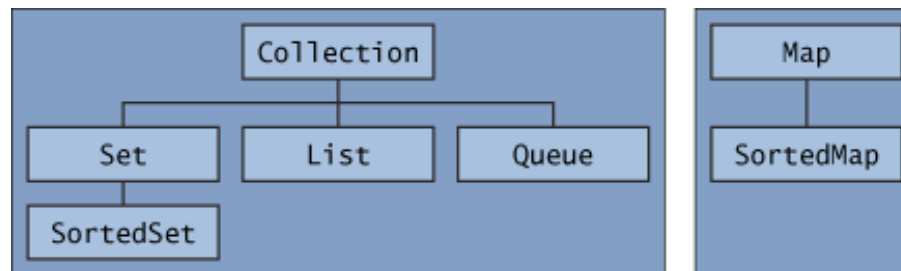
File (suite)

```
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();
}
```


Collections

Collections

- types de données
 - interfaces
 - implémentations
 - algorithmes
- Interfaces:



Collections: les interfaces

Les collections sont des interfaces génériques

- Collection<E>: add, remove, size, toArray...
 - Set<E>: éléments sans duplication
 - SortedSet<E>: ensembles ordonnés
 - List<E>: des listes éléments non ordonnés et avec duplication
 - Queue<E>: files avec tête: peek, poll (défiler), offer (enfiler)
- Map<K,V>: association clés valeurs
- SortedMap<K,V> avec clés triées

Certaines méthodes sont **optionnelles** (si elles ne sont pas implémentées UnsupportedOperationException).

En plus:

- Iterator<E>: interface qui retourne successivement les éléments next(), hasNext(), remove()
- ListIterator<E>: itérateur pour des List, set(E) previous, add(E)

Collection

```
public interface Collection<E> extends Iterable<E> {
    // operations de base
    int size();
    boolean isEmpty();
    boolean contains(Object element);
    boolean add(E element);           //optionnel
    boolean remove(Object element); //optionnel
    Iterator<E> iterator();

    // operations des collections
    boolean containsAll(Collection<?> c);
    boolean addAll(Collection<? extends E> c); //optionnel
    boolean removeAll(Collection<?> c);       //optionnel
    boolean retainAll(Collection<?> c);       //optionnel
    void clear();                               //optionnel

    // Array
    Object[] toArray();
    <T> T[] toArray(T[] a);
}
```

Collection

□ Les collections sont génériques

□ Parcours:

■ On peut parcourir les éléments par « for »:

```
for (Object o : collection)
    System.out.println(o);
```

■ Ou avec un Iterator:

```
static void filter(Collection<?> c) {
    for (Iterator<?> it = c.iterator();
        it.hasNext();)
        if (!cond(it.next()))
            it.remove();
}
```



Collection

- On peut convertir une collection en tableau
 - En tableaux de Object
 - En tableaux d'objet du type paramètre de la collection
- Il existe aussi une classe Collections qui contient des méthodes statiques utiles

Set

- Interface pour contenir des objets différents
 - Opérations ensemblistes
 - SortedSet pour des ensembles ordonnés
- Implémentations:
 - HashSet par hachage (performances)
 - TreeSet arbre rouge-noir
 - LinkedHashSet ordonnés par ordre d'insertion

Set

```
public interface Set<E> extends Collection<E> {
    // Basic operations
    int size();
    boolean isEmpty();
    boolean contains(Object element);
    boolean add(E element);           //optional
    boolean remove(Object element); //optional
    Iterator<E> iterator();

    // Bulk operations
    boolean containsAll(Collection<?> c); // sous-ensemble
    boolean addAll(Collection<? extends E> c); //optionnel- union
    boolean removeAll(Collection<?> c);      //optionnel- différence
    boolean retainAll(Collection<?> c);      //optionnel- intersection
    void clear();                             //optionnel

    // Array Operations
    Object[] toArray();
    <T> T[] toArray(T[] a);
}
```


Exemple:

```
public static void chercheDoublons(String ... st){
    Set<String> s = new HashSet<String>();
    for (String a : st)
        if (!s.add(a))
            System.out.println("Doubleton: " + a);

    System.out.println("il y a "+s.size() + " mots différents: " + s);
}

public static void chercheDoublonsbis(String st[]){
    Set<String> s=new HashSet<String>();
    Set<String> sdup=new HashSet<String>();
    for(String a :st)
        if (!s.add(a))
            sdup.add(a);
    s.removeAll(sdup);
    System.out.println("Mots uniques:      " + s);
    System.out.println("Mots dupliqués:  " + sdup);
}
```



Lists

- En plus de Collection:
 - Accès par position de l'élément
 - Recherche qui retourne la position de l'élément
 - Sous-liste entre deux positions
- Implémentations:
 - ArrayList
 - LinkedList

List

```
public interface List<E> extends Collection<E> {
    // Positional access
    E get(int index);
    E set(int index, E element); //optional
    boolean add(E element); //optional
    void add(int index, E element); //optional
    E remove(int index); //optional
    boolean addAll(int index,
        Collection<? extends E> c); //optional

    // Search
    int indexOf(Object o);
    int lastIndexOf(Object o);

    // Iteration
    ListIterator<E> listIterator();
    ListIterator<E> listIterator(int index);

    // Range-view
    List<E> subList(int from, int to);
}
```



Itérateur pour listes

```
public interface ListIterator<E> extends Iterator<E> {  
    boolean hasNext();  
    E next();  
    boolean hasPrevious();  
    E previous();  
    int nextIndex();  
    int previousIndex();  
    void remove(); //optional  
    void set(E e); //optional  
    void add(E e); //optional  
}
```

Exemple

```
public static <E> void swap(List<E> a, int i, int j) {
    E tmp = a.get(i);
    a.set(i, a.get(j));
    a.set(j, tmp);
}
public static void melange(List<?> list, Random rnd) {
    for (int i = list.size(); i > 1; i--)
        swap(list, i - 1, rnd.nextInt(i));
}
```

Suite...

```
public static <E> List<E> uneMain(List<E> deck, int n) {
    int deckSize = deck.size();
    List<E> handView = deck.subList(deckSize - n, deckSize);
    List<E> hand = new ArrayList<E>(handView);
    handView.clear();
    return hand;
}
public static void distribuer(int nMains, int nCartes) {
    String[] couleurs = new String[]{"pique", "coeur", "carreau", "trèfle"};
    String[] rank = new String[]
    {"as", "2", "3", "4", "5", "6", "7", "8", "9", "10", "valet", "dame", "roi"};
    List<String> deck = new ArrayList<String>();
    for (int i = 0; i < couleurs.length; i++)
        for (int j = 0; j < rank.length; j++)
            deck.add(rank[j] + " de " + couleurs[i]);
    melange(deck, new Random());
    for (int i=0; i < nMains; i++)
        System.out.println(uneMain(deck, nCartes));
}
```

Map

- Map associe des clés à des valeurs
 - Association injective: à une clé correspond exactement une valeur.
 - Trois implémentations, comme pour set
 - HashMap,
 - TreeMap,
 - LinkedHashMap
 - Remplace Hash

Map

```
public interface Map<K,V> {
    // Basic operations
    V put(K key, V value);
    V get(Object key);
    V remove(Object key);
    boolean containsKey(Object key);
    boolean containsValue(Object value);
    int size();
    boolean isEmpty();
    // Bulk operations
    void putAll(Map<? extends K, ? extends V> m);
    void clear();
    // Collection Views
    public Set<K> keySet();
    public Collection<V> values();
    public Set<Map.Entry<K,V>> entrySet();
    // Interface for entrySet elements
    public interface Entry {
        K getKey();
        V getValue();
        V setValue(V value);
    }
}
```


Exemples

```
public static void mapFreq(String ... t) {
    Map<String, Integer> m = new HashMap<String,Integer>();
    for (String a : t) {
        Integer freq = m.get(a);
        m.put(a, (freq == null) ? 1 : freq + 1);
    }
    System.out.println("Il y a: " + m.size() +
        " mots différents:\n"+m);
}
// ordre arbitraire
```

Exemples

```
public static void mapFreq(String ... t) {
    Map<String, Integer> m = new TreeMap<String,Integer>();
    for (String a : t) {
        Integer freq = m.get(a);
        m.put(a, (freq == null) ? 1 : freq + 1);
    }
    System.out.println("Il y a: " + m.size() +
        " mots différents:\n"+m);
}
// ordre arbitraire
```

Exemples

```
public static void mapFreq(String ... t) {
    Map<String, Integer> m = new LinkedHashMap<String,Integer>
        ();
    for (String a : t) {
        Integer freq = m.get(a);
        m.put(a, (freq == null) ? 1 : freq + 1);
    }
    System.out.println("Il y a: " + m.size() +
        " mots différents:\n"+m);
}
// ordre arbitraire
```

Queue

- Pour représenter une file (en principe FIFO):
 - Insertion: offer -add
 - Extraction: poll - remove
 - Pour voir: peek -element
 - (retourne une valeur - exception
- PriorityQueue implémentation pour une file à priorité



Interface Queue

```
public interface Queue<E> extends  
    Collection<E> {  
    E element();  
    boolean offer(E e);  
    E peek();  
    E poll();  
    E remove();  
}
```

Exemple

```
public static void compteur(int n)
    throws InterruptedException {
    Queue<Integer> queue = new
        LinkedList<Integer>();
    for (int i = n; i >= 0; i--)
        queue.add(i);
    while (!queue.isEmpty()) {
        System.out.println(queue.remove());
        Thread.sleep(1000);
    }
}
```

Exemple

```
static <E> List<E> heapSort(Collection<E> c) {  
    Queue<E> queue = new PriorityQueue<E>(c);  
    List<E> result = new ArrayList<E>();  
    while (!queue.isEmpty())  
        result.add(queue.remove());  
    return result;  
}
```

Des implémentations

- HashSet<E>: implémentation de Set comme table de hachage. Recherche/ ajout suppression en temps constant
- TreeSet<E>: SortedSet comme arbre binaire équilibré $O(\log(n))$
- ArrayList<E>: liste implémentée par des tableaux à taille variable accès en $O(1)$ ajout et suppression en $O(n-i)$ (i position considérée)
- LinkedList<E>: liste doublement chaînée implémente List et Queue accès en $O(i)$
- HashMap<K,V>: implémentation de Map pas table de hachage ajout suppression et recherche en $O(1)$
- TreeMap<K,V>: implémentation de SortedMap à partir d'arbres équilibrés ajout, suppression et recherche en $O(\log(n))$
- WeakHashMap<K,V>: implémentation de Map par table de hachage
- PriorityQueue<E>: tas à priorité.

Comparaisons

- Interface Comparable<T> contient la méthode
 - public int compareTo(T e)
 - "ordre naturel"
- Interface Comparator<T> contient la méthode
 - public int compare(T o1, T o2)



Quelques autres packages

- System méthodes static pour le système:
 - entrée-sorties standard
 - manipulation des propriétés systèmes
 - utilitaires "Runtime" `exit()`, `gc()` ...



Runtime, Process

- Runtime permet de créer des processus pour exécuter des commande: `exec`
- Process retourné par un `exec` méthodes
 - `destroy()`
 - `exitValue()`
 - `getInputStream()`
 - `getOutputStream()`
 - `getErrorStream()`



Exemple

- exécuter une commande (du système local)
 - associer l'entrée de la commande sur System.in
 - associer la sortie sur System.out.

Exemple

```
class plugTogether extends Thread {
    InputStream from;
    OutputStream to;
    plugTogether(OutputStream to, InputStream from ) {
        this.from = from; this.to = to;
    }
    public void run() {
        byte b;
        try {
            while ((b= (byte) from.read()) != -1) to.write(b);
        } catch (IOException e) {
            System.out.println(e);
        } finally {
            try {
                to.close();
                from.close();
            } catch (IOException e) {
                System.out.println(e);
            }
        }
    }
}

I/O package
}
```

Exemple suite

```
public class Main {
    public static Process userProg(String cmd)
    throws IOException {
        Process proc = Runtime.getRuntime().exec(cmd);
        Thread thread1 = new plugTogether(proc.getOutputStream(), System.in);
        Thread thread2 = new plugTogether(System.out, proc.getInputStream());
        Thread thread3 = new plugTogether(System.err, proc.getErrorStream());
        thread1.start(); thread2.start(); thread3.start();
        try {proc.waitFor();} catch (InterruptedException e) {}
        return proc;
    }
    public static void main(String args[])
    throws IOException {
        String cmd = args[0];
        System.out.println("Execution de: "+cmd);
        Process proc = userProg(cmd);
    }
}
```