Les threads

- Threads et processus
- Création de threads en Java
- Synchronisation
- Thread et JavaFX

Threads et Processus : généralités

Un processus est un programme en cours d'exécution.

Chaque processus dispose d'un espace mémoire :
- pour le code
- pour les données
- pour les piles d'exécution (variables locales et adresses de retour des fonctions)

Chaque processus est composé de plusieurs threads (au moins un créé au moment de la création du processus).
Les threads partagent le même espace mémoire (code et données) et dispose chacun d'une pile d'exécution.

Les threads sont exécutés à tour de rôle sous le contrôle du système d'exploitation.

Le processus ou thread choisi est choisi parmi les plus prioritaires pour une durée d'exécution fixée par le système d'exploitation.

Threads et Processus avec java

A la machine virtuelle correspond 1 processus (créé au lancement de la machine virtuelle).
Le thread Main correspond à l'exécution de la méthode main.

Autres threads :
- le garbage collector
- JAVAFX Application thread

La classe java.lang.Thread permet de créer des threads

Les threads : un premier exemple

On souhaite pouvoir afficher une date régulièrement sur la sortie standard pendant une durée déterminée.

```
public static void main(String [ ] args) throws Exception{
  /** creation d'un thread permettant l'affichage toutes les 60 secondes
     * de la date sur la sortie standard pendant 1 heure */
 Thread threadDate = new ThreadDate (60,3600);
  /** demarrer ce thread*/
 threadDate.start( );

 // ici les instructions de notre programme qui seront exécutées "en parallèle"
Compte c1 = new Compte("Dupond",200,200);
...
}
```

Les threads : un premier exemple

```
public class ThreadDate extends Thread{
  private int periode;private Date dateLimite;
  /** crée un thread d'affichage de la date toutes les 'periode' secondes
pendant 'duree' secondes*/
  public ThreadDate (int periode,int duree){
this.periode=periode;
this.dateLimite=new Date(new Date( ).getTime( )+duree*1000);}
  /** le travail à effectuer par ce thread :
affichage à intervalle régulier d'une date jusqu'à une date limite*/
  public void run( ){
      boolean continuer=true;
while (continuer) {
      date dateActuelle = new Date( );
      if (dateActuelle.after(dateLimite))
            continuer=false;
        else {System.out.println("Il est "+ dateActuelle);
              try {Thread.sleep(periode *1000);}
                   catch (InterruptedException exp){
                      exp.printSTackTrace( );}
             }
}}}
```

Création d'un thread : la méthode start

La méthode start de la classe Thread provoque :

1. La création par le système d'exploitation d'un thread
2. L'exécution par ce thread de la méthode run

```
/**
   * Causes this thread to begin execution;
   *  the Java Virtual Machine calls the run method of this thread.
   * The result is that two threads are running concurrently: the
   * current thread (which returns from the call to the start method)
   * and the other thread (which executes its run method).
   *
   * @exception  IllegalThreadStateException  if the thread was already
   *                started.
*/
   public synchronized native void start( );
```

Quelques fonctions de la classe Thread

/** crée un objet thread */
public Thread ( ){…}
/**crée un objet thread qui consistera en l'exécution de la méthode run
du runnable passé en paramètre*/
public Thread (Runnable runnable){…}

/** la méthode exécutée par le thread*/
public void run( ) { … }
/** la méthode créant le thread */
public void start( ) { … }
/** teste si ce thread est vivant
un thread est vivant si la méthode run est lancée mais non terminée */
public boolean isAlive( ){...}
/** waits for this thread to die */
public final void join( ) throws InterruptedException {…}
/** le thread devient inactif pendant la durée indiquée */
public static void sleep(long milliseconds) throws InterruptedException{…}

Les threads : méthode start et méthode run

La méthode start crée un thread
et lance l'éxécution de la méthode run dans ce thread.

Le thread se termine lorsque la méthode run se termine

La méthode run ne doit pas être appelée directement :
c'est le rôle de la méthode start

Si la méthode run est appelée directement, dans ce cas,
elle est exécutée dans le même thread et il n'y a pas de nouveau thread créé !

Les threads : un deuxieme exemple

```
public static void main(String [ ] args)throws Exception{
 Thread ta = new ThreadAfficheLettre ('a',2000);
 Thread tb = new ThreadAfficheLettre ('b',2000);
 ta.start( );
 tb.start( );
}
```

```
aaaaaaaaaaaabbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
bbbbbbbbbbbbbbbbaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaabbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
bbbbbbbbbbbbbbbbaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaabbbbbbbbbbbbbbbbbbb
bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
```

Les threads : un deuxieme exemple

```
public class ThreadAfficheLettre  extends Thread{
/** la lettre à afficher*/
private char lettre;
/** le nombre d'affichage à effectuer*/
private int lim;
/** crée un thread affichant 'lim' fois la lettre 'lettre'*/
public ThreadAfficheLettre (char lettre,int lim){
      this.lettre=lettre;
      this.lim=lim;
}
/** le travail à effectuer par ce thread */
public void run( ){
      int cpt=0;
      while (cpt<=lim){
            cpt++;
            System.out.print(lettre); System.out.flush( );
            }
       } }
```

Les threads : fonctionnement

La méthode start de la classe Thread est appelée pour lancer effectivement le thread.

```
Thread ta = new ThreadAfficheLettre ('a',2000);
Thread tb = new ThreadAfficheLettre ('b',2000);
ta.start( );
tb.start( );
```

Le thread système n'est effectivement créé qu'à l'appel de la méthode start

La méthode start se traduit par la création d'un thread par le système d'exploitation et par l'exécution dans ce thread de la méthode run appliquée à l'objet thread java.

```
/** le travail à effectuer par ce thread */
  public void run( ){
int cpt=0;
while (cpt<=lim){
            cpt++;
            System.out.print(lettre); System.out.flush( );
            } }
```

les deux façons de procéder pour lancer un thread

Deux constructeurs fréquemment utilisés

Le premier est celui qui est appelé par les constructeurs des classes dérivées et que nous avons utilisé implicitement dans les exemples précédents

```
/** celui qui est utilisé lors de la création d'une instance indirecte de la classe Thread
  * La méthode start exécute dans un nouveau thread
  *                          la méthode run redéfinie de la classe dérivée appliquée à ce thread *
public Thread( ) {...}
```

Le second qui prend en paramètre un Runnable

```
/** on peut aussi créer un Thread en passant en paramètre un Runnable
   * la méthode start exécute dans un nouveau thread la méthode run de 'target' */
public Thread(Runnable target) {…}
```

```
/** Runnable est une interface qui ne déclare qu'une seule méthode */
public interface Runnable {public void run( );}
```

Les threads : autre façon de procéder

```
public static void main(String [ ] args)throws Exception{
    Runnable ra = new RunnableAfficheLettre('a',200);
    Runnable rb = new RunnableAfficheLettre('b',200);
    Thread tha = new Thread(ra);tha.start( );
    Thread thb = new Thread(rb); thb.start( );
    }
```

```
public class RunnableAfficheLettre implements Runnable{
private char lettre; private int lim;
public RunnableAfficheLettre (char lettre,int lim){
      this.lettre=lettre;this.lim=lim;
}
public void run( ){
            int cpt=0 ;
            while (cpt<=lim){
                  cpt++;
                  System.out.print(lettre); System.out.flush( );
                  } }
```

Ceci permet de lancer des threads avec des objets qui ne sont pas des instances de Thread

La méthode run de la classe java.lang.Thread

```
/**
    * If this thread was constructed using a separate Runnable run object,
    *  then that Runnable object's run method is called;
    * otherwise, this method does nothing and returns.
    *
    * Subclasses of Thread should override this method.
    *
*/
   public void run( ) {
     if (this.target != null) {
           this.target.run( );
     }
   }
```

Remarque : la donnée membre target correspond à l'objet Runnable passé en paramètre à la construction

Autres méthodes de la classe Thread

```
// autres méthodes de la classe Thread

/** rend le thread actuellement en exécution */
public static Thread currentThread( ){…}

/** rend le groupe de thread auquel appartient ce thread
    (chaque thread appartient à un groupe de thread) */
public ThreadGroup getThreadGroup( ){…}

/** teste si ce thread est un daemon
  * la mv s'arrête lorsque tous les threads sont arrêtés
s'il ne reste plus que des threads daemons, ils sont arretes*/
public boolean isDaemon( ) { … }

/** teste si ce thread est vivant
     un thread est vivant si la méthode run est lancée mais non terminée */
public boolean isAlive( ){...}
```

Classes internes : exemple d'utilisation

```
Runnable runnable = new Runnable( ) {
public void run( ){
      Fenetre.this.executerTraitementLong( );
}
       }
new Thread(runnable).start( );
```

La méthode join

/ waits for this thread to die */**

public final void join( ) throws InterruptedException {…}

```
public static void main(String [ ] args)throws Exception{
Thread ta = new ThreadAfficheLettre ('a',200);
Thread tb = new ThreadAfficheLettre ('b',200);
        ta.start( );
        tb.start( );
        ta.join( );
        tb.join( );
        System.out.println("fin");
}
```

SYNCHRONISATION

- **mot clé synchronized**

- méthodes wait( ) , notify( ), notifyAll( ) de Object

synchronized

```
synchronized (list) {
    list.add("ok");
    }
```

2 blocs synchronisés sur un même objet ne peuvent être exécutés simultanément par 2 threads différents

```
synchronized (obj){    //Pose d'un verrou sur l'objet
…
…
…
}                                  // retrait du verrou
```

Le thread devant exécuter un bloc synchronisé reste en attente tant qu'il existe déjà un verrou sur l'objet concerné.

synchronized sur les méthodes

```
// Class vector
public synchronized int size( ) {…}
```

équivalent à :

```
public int size( ) {
   synchronized (this) {…}}
```

Aucun bloc synchronisé portant sur la même instance
ne peut s'exécuter dans un autre thread tant que la méthode synchronisée n'est pas terminée

synchronized : exemple

```
public void run( ){
    while (true){
        Mail mail = getMail();
        if (mail==null)
            try {Thread.sleep(1000);}
            catch (InterruptedException e) {e.printStackTrace();   }
        else sendMail(mail);
    }
/** ajoute ce mail à la liste des mails */
public synchronized void addMail(Mail mail){
    mailList.add(mail);
    }

/** renvoie un mail à envoyer , null si plus de mail à envoyer*/
public synchronized Mail getMail( ){
    if (mailList.isEmpty())
       return null;
    Mail mail = mailList.get(0);
    mailList.remove(0);
    return mail;
}
```

wait notify notifyAll

Ce sont trois méthodes de la classe Object

wait le thread courant est stoppé jusqu'à ce qu'un autre thread appelle notify ou notifyAll sur ce même objet.
wait libère le verrou

notify débloque un thread en attente

notifyAll débloque tous les threads en attente

Thread et JavaFX

Un thread spécifique (JavaFX Application Thread**) est chargé de la gestion des événements.**

TOUTES les modifications affectant un composant doivent être faite par ce thread

public class Platform {
/** Run the specified Runnable on the JavaFX Application Thread at some unspecified time in the future. This method, which may be called from any thread, will post the Runnable to an event queue and then return immediately to the caller. The Runnables are executed in the order they are posted.
@param runnable - the Runnable whose run method will be executed on the JavaFX Application Thread
**/

public static void runLater(java.lang.Runnable runnable)

Thread et JavaFX : runLater

```
Runnable runnable = new Runnable( ) {
public void run( ){
      label.setText("OK");
      }
      }
Platform.runLater(runnable );
```

runnable.run() sera exécuté plus tard par le thread javaFX

runLater : exemple

```
public class ButtonBlink extends Button implements Runnable{

  private boolean continuer=true;

public ButtonBlink( ) {
    this.setOnAction((e)->arreter());
  }

  public void run( ) {
    while (continuer){
       sleep(500);
       Platform.runLater(
                  ( )->ButtonBlink.this.setStyle("-fx-background-color:red;"));
       sleep(500);
       Platform.runLater(
                ( )->ButtonBlink.this.setStyle("-fx-background-color:green;"));
    }
  }

public void arreter( ){this.continuer=false;}
```

Méthode sleep

```
private void sleep(int milliseconds){
    try {Thread.sleep(milliseconds);}
   catch (InterruptedException e) {e.printStackTrace();}
   }
```

L'appli pour tester le bouton qui clignote

```
public class AppliButtonBlink extends Application {

    @Override
    public void start(Stage stage) {
        stage.setTitle("Blink");

        ButtonBlink button = new ButtonBlink();
        Scene scene = new Scene(button);
        new Thread(button).start();
        stage.setScene(scene);
        stage.show();
    }

    public static void main(String[] args) {
        launch(args);
    }
}
```