INFO-H-200
Programmation orientée objet

Séance d'exercices 5
Threads et Libraire javax.Swing

Université libre de Bruxelles
École polytechnique de Bruxelles

Professeur : Hugues Bersini

2014-2015

Threads

Thread

Un Thread est un processus

Le but de l'utilisation des threads est d'exécuter des opérations en parallèle.

Exemple : Un logiciel de traitement de texte vérifie l'orthographe pendant que l'utilisateur tape du texte.

- Un thread qui gère le clavier
- Un thread qui gère la correction orthographique

Pour donner une illusion de parallélisme sur un processeur, l'OS (ou la JVM) exécute les processus et threads de façon concurrent et synchronisée.

Comment faire ?

Deux manières de faire :

- Hériter de la classe Thread (peu pratique)
- Implémenter l'interface Runnable (préféré)

Une classe *thread* doit contenir une méthode *void run()* qui défini le comportement et le cycle de vie de ce thread.

Une fois la méthode *run()* terminée, le thread s'arrête et est détruit.

Pour démarrer le thread, il faut appeler sa méthode start().

Exemple

```
public class MyTimer implements Runnable{
   String string;
   int waitTime;

   public MyTimer(String string, int waitTime){
      this.string = string;
      this.waitTime = waitTime;
   }

   @Override
   public void run(){
      try{
         while(true){
            System.out.print(string);
            Thread.sleep(waitTime);
         }
      }catch(Exception e){};
   }

   public static void main(String[] args){
      Thread t1 = new Thread(new MyTimer("A",100));
      Thread t2 = new Thread(new MyTimer("B",100));
      Thread t3 = new Thread(new MyTimer("C",200));
      t1.start();
      t2.start();
      t3.start();
   }
}
```

Réservation de ressources

Comme ce n'est pas le programmeur qui gère l'ordonnancement des threads, il peut arriver que deux threads accèdent à la même ressource en même temps.

Cela peut être dangereux. Pour cela, un thread qui utilise une ressource partagée doit la réserver pour en avoir un accès exclusif.

Pour avoir un accès exclusif à un objet, ou attendre que celui-ci soit disponible :

```
synchronized(unObjet){
   // Section critique
}
```

Deadlocks

Situation d'interblocage

Exemple :

- soit deux threads t1 et t2 et deux ressources r1 et r2
- t1 a la main et bloque r1
- t2 prend la main et bloque r2
- t1 prend la main et veut bloquer r2. Il doit attendre que r2 soit libre et donc que t2 la libère
- t2 prend la main et veut bloquer r1. Il doit attendre que r1 soit libre et donc que t1 la libère.
- Le système est bloqué !

Swing

Swing

Swing est la libraire standard Java pour créer des interfaces graphiques (GUI). Doc Java

Fait partie des Java Foundation Classes (JFC) qui contiennent d'autres outils utilses aux applications graphiques :

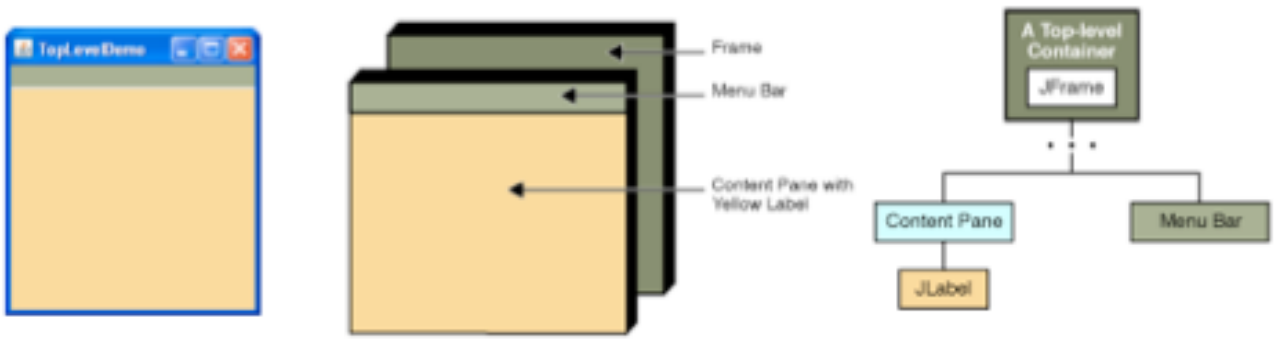
- Swing GUI Components : boutons, textes, menus, checkboxes, tableaux, boites de dialogues,...
- Java 2D API : Libraire d'affichage 2D sur laquelle se base Swing
- Synth : Librairie permettant de créer son propre "look and feel"
- Data transfert : Gestion du copier/coller, du drag and drop
- Internationalization : Gestion des différentes langues et caractères
- Accessibilité API, Undo Framework API,...

Conteneurs Swing

Un composant Swing (label, bouton,...) ne peut se trouver dans un et un seul conteneur.

Un conteneur est un composant peut contenir d'autres composants. Si le conteneur contient des conteneurs, on parle alors de "containment hierarchy".

Une fenêtre (un Top-Level container) est un conteneur et est implémentée dans la classe JFrame. Une JFrame est composée d'un titre, d'un Content Pane et éventuellement d'un Menu Bar.



Composants Swing

A l'exception des Top-level containers, toutes les classes Swing qui commencent par J descendent de la classe JComponent. (JPanel, JButton, JLabel,...)

JComponent descend de Container qui descend de Component. Un conteneur est donc un composant.

Tous les JComponent ont des méthodes communes :

- Gérer l'apparence
- Gérer l'état
- Gérer des événements
- Dessigner les composants
- Placer les composants dans des conteneurs, spécifier leur position
- ...

Exemple : HelloWorld

```
package Example;

import java.awt.*;
import javax.swing.*;

public class GUI {
   public static void main(String[] args){
      JFrame frame = new JFrame("HelloWorld"); // Nouvelle fenêtre (top level
            container)
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

      JLabel label = new JLabel("HelloWorld"); // Nouveau label jaune contenant
            le texte "HelloWorld"
      label.setOpaque(true);
      label.setBackground(Color.YELLOW);

      frame.getContentPane().add(label); // Ajout du label dans le ContentPane

      frame.pack(); // Dimensionne la frame

      frame.setVisible(true); // Affiche la fenêtre
   }
}
```

Quelques composants

- **JButton :** un bouton
- **JCheckBox :** un bouton à cocher
- **JRadioButton :** un bouton radio (doit être dans un ButtonGroup)
- **JLabel :** un label non éditable
- **JTextField :** un champ de texte
- **JTextArea :** une zone de texte (plusieurs lignes)
- **JDialog :** boîtes de dialoge (voir JOptionPane)
- **JList :** liste
- **JComboBox :** liste déroulante
- **JProgressBar :** barre de progression
- **JTable, JTree :** tableaux et arbres (voir plus loin)
- ...

Panels

On peut grouper des composants dans des JPanel (un conteneur)

Un JPanel peut avoir un layout qui définira comment seront disposés les composants à l'intérieur du conteneur

Par défaut le layout est un FlowLayout, c'est à dire que les composants seront placés les uns à la suite des autres en fonction de la taille de la fenêtre

BorderLayout permet d'ajouter des composants dans cinq zones différentes :


```
JPanel p = new JPanel(new BorderLayout());
p.add(quelqueChose, BorderLayout.LINE_END);
```

Gestion des événements

Quand on intéragit avec un élément (p.e : un bouton), un événement est déclenché.

Un gestionnaire d'événement est donc nécessaire pour y réagir.

Solution simple : implémenter l'interface ActionListener et sa méthode actionPerformed(ActionEvent e)

Exemple : Programme avec un bouton déclenchant un son lorsqu'il est cliqué

```
public class GUI{
   JButton button;

   public GUI(){
   ...
      button.addActionListener(new BeepListener());
   ...
   }
}

public class BeepListener implements ActionListener{
   public void actionPerformed(ActionEvent e){
      ...
      // Make a beep sound...
   }
}
```

Gestion des événements (suite)

Ce système est simple et flexible

Exemple : Un programme pourrait créer un listener par source d'événement ou créer un seul listener pour toutes les sources ou encore avoir plus d'un listener pour un même événement d'une même source.

Chaque événement est représenté par un objet qui donne de l'information sur l'événement et sur sa source. Les sources sont souvent des composants mais ce n'est pas toujours le cas (ex : un message reçu sur un réseau)

Types d'événements

Les composants peuvent déclencher des événements différents. Le plus simple à utiliser et le plus courant est l'événement Action. Par contre, JColorChooser peut déclencher un événement de type Change qui est un peu différent (pour lequel il faudra implémenter ChangeListener. Idem pour la souris qui peut déclencher des événements Mouse ou MouseMotion.

Lors de l'utilisation d'un composant particulier, il est donc utile de consulter sa documentation (1) et celle de chaque événement (2) qu'il peut déclencher :

1. `http://docs.oracle.com/javase/tutorial/uiswing/components/`
2. `http://download.oracle.com/javase/tutorial/uiswing/events/handling.html`

Remarques

Un listener doit pouvoir s'exécuter très rapidement car la gestion des événements et du dessin d'une application s'exécutent dans le même thread. Un listener lent pourrait donc figer l'application le temps de son exécution.

Il est plus sûr d'implémenter des listeners dans des classes package-private

Un événement est un objet qui hérite de la classe EventObject, par exemple : MouseEvent

Object getSource() d'EventObject retourner l'objet qui a déclenché l'événement. Il existe des méthodes plus spécifiques comme getComponent(), implémentée dans les événements de type ComponentEvent.

Certaines interfaces listener contiennent plus d'une méthode. Par exemple, MouseListener contient 5 méthodes : mousePressed, mouseClicked,... Ce n'est pas une mauvaise pratique de laisser certaines méthodes vides.