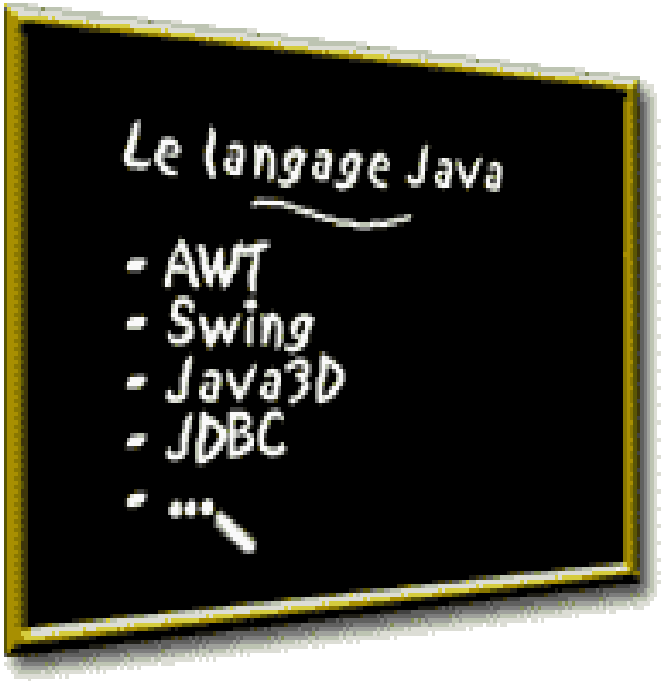
Cours 4



- Créer des animations : la classe Thread et l'interface Runnable

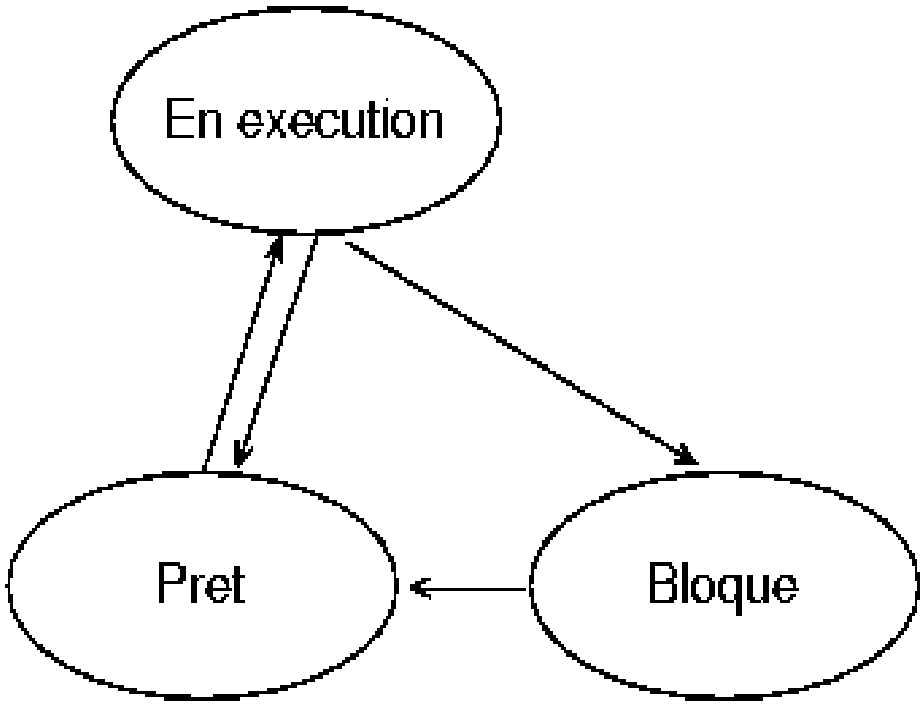
- Quelques composants supplémentaires : le choix dans une liste

JComboBox et JList

Rappel : multi-tâches

- **multi-tâches** : exécution de plusieurs processus simultanément (Windows NT, Unix...).
  - Un processus est un **programme en cours d'exécution**.
  - L'ordonnanceur distribue le temps CPU entre les processus
- Un processus peut être dans différents états.
  - En **exécution** (running) : a le processeur
  - **Prêt** : le processus est prêt à s'exécuter, mais n'a pas le processeur (occupé par un autre processus en exécution)
  - **Bloqué**

états d'un processus et transitions



Thread passe de l'état en exécution à l'état bloqué :
- S'il reçoit l'ordre de dormir
- s'il a besoin d'une ressource qui n'est pas disponible

Thread passe de l'état bloqué à l'état Prêt :
- Si son temps de sommeil est épuisé
- Si la ressource qu'il attendait s'est libérée

Rappel : multi-threads

- Un thread ("processus léger") est un processus à l'intérieur d'un processus.
- **multi-threading** : exécution à l'intérieur d'un processus de plusieurs sous-tâches (exécution concurrente de threads)
  - Un processus peut posséder plusieurs threads.
  - Les ressources allouées à un processus (temps processeur, mémoire) sont partagées entre les threads qui le composent.
  - Un processus possède au moins un thread (qui exécute le programme principal, habituellement la fonction *main()*).

Différences entre thread et processus

- Au niveau de la mémoire :
  - Les sous-processus issus d'un même processus ont leur propre espace d'adressage et doivent, pour communiquer, utiliser des moyens de communication specifiques (tubes..)
  - Les threads issus d'un même processus partagent la même zone mémoire (segments de code et de données), ce qui rend très facile (et périlleux !) la communication entre threads.

Intérêts des threads en général

- Communication très simple grâce aux données partagées
- Partager le temps alloué au processus père entre plusieurs threads, chacun d'eux exécutant une fonction précise
- Augmenter la "productivité" d'une application par l'exécution concurrente de ces threads
- Exemples
  - Un navigateur est un logiciel multi-thread : le chargement d'une page laisse la main à l'utilisateur
  - Un serveur peut répondre à des demandes de connexions de clients en créant un thread par client

Intérêt des threads pour une application graphique

- Tout programme qui s'exécute possède au moins un thread : le thread principal, celui qui exécute…
- Il existe d'autres threads en arrière-fond :
  - le thread dédié au garbage
  - un thread qui traite les événements clavier et souris
- On crée d'autres threads lorsque l 'application nécessite d 'exécuter une tâche longue et consommatrice de temps ou une tâche répétitive (Exemple : animation, gros calculs, attente de connexion, entrée/sortie (chargement d'images))

  ⇒ Créer des threads chargés de ces tâches plutôt que d 'occuper le thread principal à ces tâches

Threads en java

- En Java, les threads font partie intégrante du langage:
  - La classe Thread
  - L'interface Runnable
- Les programmes utilisant des threads sont portables d'une plate-forme à l'autre mais l'exécution peut varier (comportement de l'ordonnanceur non spécifié par Sun)

La classe Thread

- **Quelques constructeurs :**

Thread(); Thread(String nom);
Thread(Runnable cible);

- **Quelques méthodes d'instance :**

String getName();

void start(); **méthode qui met le thread à l'état prêt**

void run(); **méthode qui sera exécutée dès que le thread aura la main**

boolean isAlive();

void setPriority(int newPriority);

La classe Thread

- **Constantes de classe : permet de définir des priorités**

MIN_PRIORITY, MAX_PRIORITY, NORM_PRIORITY

- **Quelques méthodes de classe :**

static Thread currentThread(); **retourne la référence du thread qui a la main**

static void sleep(long millis) throws InterruptedException;

fait dormir le thread qui a la main

static int enumerate(Thread[] t) throws SecurityException

range dans t place tous les threads

Comment créer un thread chargé d'une tâche particulière

Deux méthodes

- Sous-classer la classe `Thread`
- Utiliser l'interface `Runnable` et faire que la classe souhaitant "être animée" implémente cette interface

Méthode 1 : sous-classer la classe Thread

- Créer une classe qui hérite de la classe Thread.

```
class MonThread extends Thread    {
   public void run {
   ... mettre ici le code à exécuter, c'est à dire l'activité que le thread doit avoir....
   }}
```

- L'utiliser :

```
MonThread th = new MonThread();
th.start();
```

Méthode 2: implémenter l'interface Runnable

- Faire que la classe souhaitant "être animée" implémente l'interface Runnable (dont la seule méthode est run).

```
class MaClasse extends   ...          implements
   Runnable {
public void run {
... code à exécuter par le thread ... ......
   }
}
```

- L'utiliser :

```
MaClasse mc = new MaClasse(…);
Thread th= new Thread(mc);
th.start();
```

Dans les deux cas :

- La tâche à effectuer par le thread est implémentée dans la méthode run()
- Pour lancer (mettre à l'état prêt) un thread, lui adresser la méthode `start`
- Le thread s'arrête quand il termine sa méthode run(). La tâche correspondante dans le SE disparaît alors.
- Un thread doit mourir "naturellement" :

==> On ne peut pas l'arrêter en lui adressant une méthode particulière (la méthode `stop` est deprecated)

==> si on veut l'arrêter "de l'extérieur" :

- programmer la méthode run comme une boucle qui tourne tant qu'un indicateur est vrai
- positionner l'indicateur à faux quand nécessaire

Commentaires sur la 2ème méthode

- Si un thread est construit avec comme cible un objet Runnable, le code à exécuter par le thread est le code indiqué dans la méthode run de la cible.

```
MaClasse mc = new MaClasse(...);
Thread th= new Thread(mc);
th.start();  // th est prêt à exécuter le code de la méthode run décrit dans la cible
                    (ici dans la classe MaClasse)
```

- **Intérêt de cette deuxième méthode :**

la classe MaClasse peut hériter d'une autre classe

exemple : class MaClasse extends JFrame implements Runnable

Application graphique avec Thread : un canevas d'utilisation de Runnable

```
public class MaClasse extends J..... implements
    Runnable {
// variables membres
    private Thread tache;
    private boolean vivant;

    public void run()  {
        while (vivant) ....{
                ...}
    }
+

+
```

Un exemple complet: **fenêtre affichant un texte caractère par caractère**

Démo

Analyse

- Besoin d'un seul thread chargé de l'affichage caractère par caractère d'un texte

- Besoin d'une sous-classe de JFrame avec un JTextArea en zone centrale

==> choisir la deuxième méthode

```
public class FenProgressTexteRunnable extends JFrame
    implements Runnable
```

Plus de détails sur le thread

- Quelle est l'activité du thread ?
  - afficher dans la zone de texte prévue le texte lettre après lettre.
- Quand doit-il arrêter ?
  - le texte est complètement affiché
- Quand doit-il être lancé ?
  - Quand la fenêtre s'ouvre au départ

```
public class FenProgressTexteRunnable extends JFrame implements
Runnable    {
  private JTextArea zoneTexte;
  private String texte;
  private int indText;
  private Thread thAffiche;

public FenProgressTexteRunnable(String title, String text, int w, int h)
{
    super(title);
    texte= text;
   indText = 0;
   this.initialise();
    this.setSize (w, h);

   this.lanceThread();

    this.show();
    }

public void initialise()  {
   zoneTexte=new JTextArea();
```

```
public void lanceThread() {
    if (thAffiche==null) {thAffiche=new
Thread(this);
                          thAffiche.start();
                        }
}
```

```
public void run() {
   while (indText < texte.length() ) {
          zoneTexte.append(texte.charAt(indText)+"");
          indText++;
          try {Thread.sleep(100);}
          catch(InterruptedException e)
{System.out.println("pb dans sleep");}
          }   // fin du while
System.out.println("mort de sa belle mort");
thAffiche=null;
}
public static void main (String args[]) {
String texte="L  'IUT, c  'est SUPER !!! En java graphique, on s  'amuse avec
le jeu de la vie ...\n";
new FenProgressTexteRunnable("Fenêtre texte lettre à lettre", texte,
600,400);
}
```

Remarques

- Le thread est mis à l'état prêt dans le constructeur de la fenêtre

  lanceThread()

- Dès qu'il a la main, la méthode run s'exécute et le thread commence sa boucle d'activité :

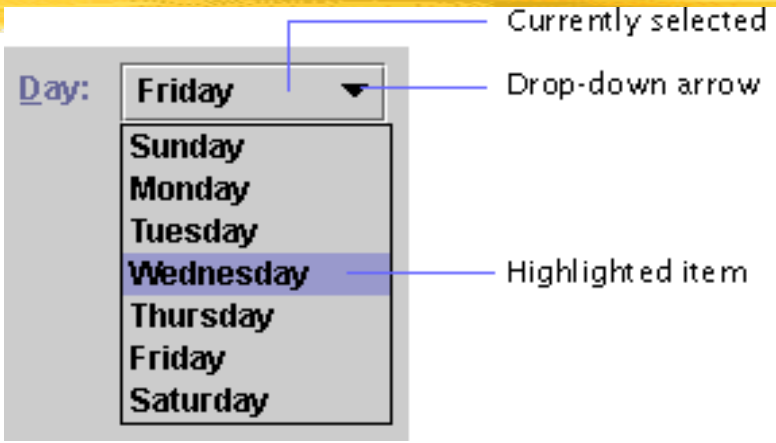
  while (indText < texte.length())

- Le processeur peut lui être repris quand il dort
- Quand il aura de nouveau la main, il continuera là où il s'était arrêté
- Le thread meurt quand il aura fini sa méthode run :
  - indText== texte.length() : tout le texte a été affiché
- On le met à null pour le faire disparaître en tant qu'objet java

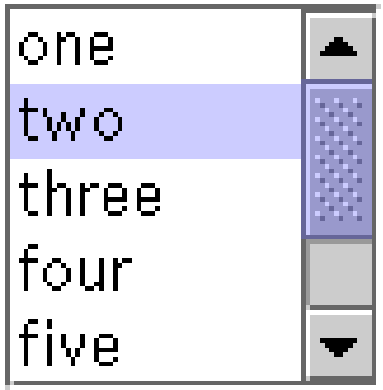
Les composants permettant un choix dans une liste

Plusieurs classes permettant un choix dans une liste

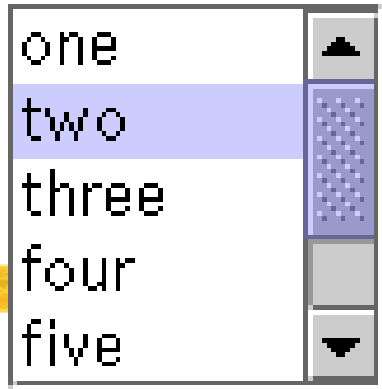
- ComboBox non éditable



- ComboBox éditable
- JList



La classe JList



- JList : composant permettant de choisir dans un ensemble connu de données
- Ces données peuvent être mémorisées
  - dans un tableau : constructeur JList(Object[] listData)

```
String[] data = {"one", "two", "three", "four", "five",   };
JList list = new JList(data);
```

  - dans un vecteur : constructeur JList(Vector listData)

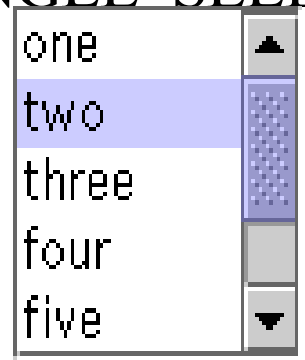
```
Vector v=new Vector(); v.addElement("one  " ); ...
.
JList list = new JList(v);
```

Sélectionner dans une liste

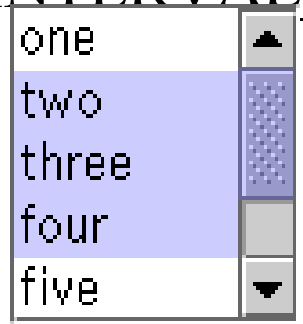
- On peut sélectionner un ou plusieurs items dans une liste suivant la valeur d 'un mode de sélection,

  constante de classe de ListSelectionModel

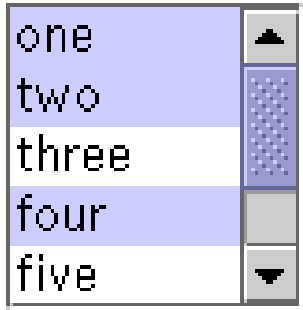
- SINGLE_SELECTION



SINGLE_INTERVAL_SELECTION



MULTIPLE_INTERVAL_SELECTION



Sélectionner dans une liste

- Par défaut, plusieurs sélections possibles
- Pour changer le mode de sélection :

```
list.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
```

- Pour reconnaître un élément sélectionné par l 'utilisateur :
  - Lier la JList à un écouteur d'événement de sélection

    interface ListSelectionListener
  - ListSelectionListener : interface qui a une seule méthode :

```
public void valueChanged(ListSelectionEvent e)
```

Récupérer les items sélectionnés dans une liste

- **Méthode de JList pour récupérer les indices**

  int getSelectedIndex()

  retourne le premier index sélectionné, -1 si aucune sélection

  int[] getSelectedIndices()

  retourne un tableau contenant les indices sélectionnés dans l 'ordre croissant

- **Méthode de JList pour récupérer les objets**

  Object getSelectedValue()

  public Object[] getSelectedValues()