4. Extensions de la programmation objet

- Introduction
  - Le projet comme clef de l'approche du génie logiciel
    - conception + programmation + BD vu comme un tout
      - outils intégrés
  - L'intégration des concepts nouveaux de la programmation
    - traitement des exceptions, encapsulation...
  - L'évolution du matériel
    - Données volatiles et persistantes
    - Concepts distribués sur les réseaux
    - Programmation multi-processeurs
      - BDOO, JAVA-net, //isme
  - La POO comme suggestion de nouvelles approches
    - organisation des classes (classes contextuelles), programmation visuelle...

- A. Le multi traitement
  - Introduction
    - le mono tache
      - Les ressources (mémoire et processus) de la machine sont dédiées à l'exécution d'un unique programme (tache)
    - le multi tache
      - chaque programme (tache Ti) se voit dédié une partie des ressources
        - une partie de la mémoire pour le programme et ses données Ti
        - un temps d'exécution fourni par intermittence par le système
    - le multi processus
      - un même programme peut être utilisé par plusieurs de façon autonome
        - une partie de la mémoire stocke le programme
        - chaque mémoire nécessaire pour les données de chacun (processus) est dédiée par le système
        - un temps d'exécution est fourni par intermittence par le système à chaque processus

- 1. Qu'est ce que le multi traitement (multi processus léger, multi threads)
  - Définition:
    Le *multi traitement* permet à plusieurs instances d'exécution (dites threads) d'un programme d'exister.
    - Tous les threads partagent le même espace mémoire (programme et données)
    - Le programme principal est aussi un thread
  - Fonctionnement
    - A un moment donné, chaque thread a un statut parmi les suivants: *prêt*, *en Sommeil*, *en attente* de notification, *bloqué* par une E/S (attend une Entrée/Sortie)
    - Un et un seul thread parmi les thread prêts est *en cours d'exécution* (c'est le système qui choisit lequel)
  - La classe thread
    - tout objet O de la classe `thread` devient un thread si il est démarré
    - `start()` : démarre un thread (prêt); puis appel méthode *`run`*`()` de O
    - `sleep()` : met la thread en sommeil (*sleep(MiliSec)*)
  - Programmer = une classe C sous classe de thread, [avec une methode *`run()`*`];`
  - Exécuter = créer O de C, puis `O<=start()`

- 2. Interruption d'un thread (JAVA)
  - Seul le thread a droit de s'interrompre
  - Classiquement
    - A la fin du programme `run()`, le thread disparaît (mais l'objet reste)
  - Une **demande** d'interuption (d'un autre thread) est possible
    - Dans Thread
      - `O.interrupt()` met un indicateur d'interruption (demande d'arrêt) dans le thread O
      - `O.isInterrupted`() dit si le thread O a l'indicateur interrupt=vrai
    - `Sleep` (et `wait`) déclenche l'erreur *`InterruptedException`* si `this.isInterrupted() =vrai`

- 3. Partage de données (exclusion mutuelle)
  - le principe du sémaphore
    - l ’accès à un objet o est verrouillé durant un traitement
    - à la fin du traitement l ’accès à O est rendu public
  - synchronized un objet
    - synchronized(o) {…}
      aucune instruction synchronisée pour o ne peut être exécutée avant la fin de l ’instruction
  - synchronized une méthode
    - methode synchronized M(..) {…}
      aucune instruction pour self ne peut être exécutée avant la fin de l ’application de la méthode

- Attente et notification (mettre après partage des données) cf. Delannoy
  - Soit o l'objet synchronisé
    - *wait*(): met en attente la thread, jusqu'à sa *notification*
    - notifyAll() : notifie à tous les thread en attente qu'ils peuvent reprendre la main

```
•public class ReserveAjoutPuise {
  public static void main (String args[])
    {Reserve r = new Reserve () ;
    Ajout ta1=new Ajout(r,100); Ajout ta2=new Ajout (r,50) ;
    Puise tp = new Puise(r, 300);
    System.out.println ("entree pour arreter ") ;
    ta1.start () ; ta2.start () ; tp.start () ;
    Clavier.lireString() ;
    ta1.interrupt (); ta2.interrupt (); tp.interrupt () ;}}
```

```
public class ReserveAjoutPuise {

  public static void main (String args[]) {Reserve r = new Reserve ();
   Ajout ta1=new Ajout(r,100);        Ajout ta2=new Ajout (r,50); Puise
   tp=new Puise(r, 300); System.out.println ("entree pour arreter ") ;
   ta1.start () ; ta2.start () ; tp.start (); Clavier.lireString() ;
   ta1.interrupt (); ta2.interrupt (); tp.interrupt () ;}}


class Reserve extends Thread {private int stock = 500 ;//stock initial
   public synchronized void puise (int v) throws InterruptedException
      {if (v <= stock) stock -= v ; else wait() ;} }
   public synchronized void ajoute (int v){stock += v ; notifyAll() ;}}

class Ajout extends Thread{private int vol; private Reserve r ;
public Ajout (Reserve r, int vol){this.vol = vol; this.r = r ;}
public void run () {try  {while (!interrupted())
                              {r.ajoute (vol) ;
                              sleep () ;}}
                    catch (InterruptedException e) {}}}

class Puise extends Thread{private int vol ; private Reserve r ;
public Puise (Reserve r, int vol){this.vol = vol; this.r = r;}
public void run () {try {while (!interrupted())
                              r.puise (vol) ; sleep () ;}}
                    catch (InterruptedException e) {};}}
```

- 4. Partage des ressources calcul (priorités)
  - si n threads et m>=n processeurs, chaque thread est affecté à un processeur
  - si n threads et m<n processeurs, les threads vont s ’exécuter de façon concurrente
    - on peut alors affecter une priorité [0,infini] à un thread
    - infini: toutes les ressources sont affectées au thread
    - 0 aucune
  - Définition
    le vieillissement est un processus qui permet à un thread d ’augmenter sa priorité au fure et à mesure du temps qui passe
  - En JAVA:
    - setPriority(X), getPriority()
    - Thread.yield() : met le thread en cours d'exécution en pret
    - Principes
      - L'environnement choisi un thread parmi ceux du niveau le plus prioritaire
      - La répartition équitable du temps entre thread n'est pas garantie (l'usage de sleep peut etre une solution)
      - Eviter d'utiliser les priorités si on veut du code portable

5. Codage

- quand on compile en java mettre l argument `-native` pour traiter les threads.

– Interface Runnable (JAVA)

- Une classe `CR` implémentant `Runnable` doit avoir une méthode `run()`
- Un objet O d'une telle classe peut avoir un thread associé to par l'instruction suivante: `Thread to = new Thread(O)`, l'appel au `run` de `CR` se fait par `to.start()`

- ```
  class Compte extends thread {  … Compte(Sring N) …
  run() {… };}
  Compte C1, C2;
  C1 = new compte(« Dupont »); C1.start();
  C2 = new compte(« Durant »); C2.start();
  ```

- ```
  class Compte extend ProduitFinancier implements Runnable { …
  Compte(Sring N) …
  run() {… };}
  Compte C1, C2; C1 = new compte(« Dupont »);
  C2 = new compte(« Durant »);
  Thread t1 = new Thread(C1); t1.start();
  Thread t2 = new Thread(C2); t2.start();
  ```

- Les threads démons (par opposition à la catégorie des threads utilisateurs)
  - Dans classe Thread: methode setDaemon(true), à appliquer avant le start
  - Propriété des threads démons: Dès que les seuls threads en exécution sont des démons, ils sont tous arrêtés (et donc le programme est fini)
  - Règles
    - Par défaut, un thread est créé de la catégorie du thread qui le crée

- C++ compatible avec multi-thread, mais à implémenter

– 6. Extensions
  - accès lecture, mais pas écriture
  - blocage de certains champs de l ’ objet

B. Le point sur des notions nouvelles

0. Les classes internes (rappel)

- Principe
  - une classe contenue dans une autre classe
- Définition
  - une *classe interne CI* (par opposition à classe globale) est une classe déclarée dans une autre classe C
- Règles
  - Seul C a accès à CI.
  - La création d ' une instance de CI ne peut se faire que dans une méthode de C
  - l ' instance de CI peut avoir accès aux champs de l ' objet O (de C) qui l ' a créée.
- Intérêts
  - une classe interne CI est invisible aux autres classes que C
  - une classe interne CI peut accéder aux champs et méthodes de sa surclasse

- 1. Les classes internes anonymes
  - Principe: une classe interne dont on se sert qu'une fois pour créer une instance n'a pas besoin d'être nommé
  - Syntaxe:
    ```
    new superC() {code la classe anonyme qui hérite (resp. implémente) la classe (resp. interface) superC}
    ```

– 2. Les classes de méthode

- Principe
  définir une classe liée à une méthode
- Définition
  une *classe de méthode* CM est une classe déclarée dans une méthode m.
- Règles
  - l ’instance de cette classe a accès aux informations de m (les variables locales de m) qui l ’a créée
  - Cet accès aux variables locales est limité à la lecture.
- Définition
  une *classe contextuelle* est une classe interne ou une classe de méthode

– 3. La Persistance

- Définition
  La *persistance* est le fait de pouvoir conserver un objet même après l ’exécution du programme.

- Mise en œuvre
  La technique classique dite de persistance par attachement utilisée en LOO, et de déclarer un objet *persistent* lors de sa création.
  Cet objet est alors persistant ainsi que tous les objets qu’il référence.

- Les langages
  O2 permet cela (les BDOO);
  java permet de sauver les objets dans un flux (si on met *transiant* devant le nom d un attribut; l attribut n ’est pas sauvegardé).

– 4. La Mutation

- Définition
  La *mutation* consiste à changer la classe d ’origine de l ’instance; elle peut s ’appliquer à des objets ayant des classes avec des champs identiques.

- # C. **La gestion des propriétés (P-listes)**
- Idée: Les listes de propriétés appelés P-listes (ou P-list) forment un type de structure de données qui permet de
  - stocker des informations de nature sémantique, comme des listes de propriétés, qui sont des données d'objets
  - un accès rapide en mémoire
  - un accès de n'importe où dans le code
- Principes
  - une P-liste associe à un *symbole S*, une *liste de propriétés L*
  - dans cette liste de propriétés L figure autant de fois qu'il y a de propriétés Pi
    - le nom de la propriété Pi suivie par sa valeur Vali
- Fonctions utiles
  - Associer à un symbole S une nouvelle propriété Pi et sa valeur Vali
  - obtenir pour un symbole S sa liste de propriétés avec leurs valeurs associées
  - obtenir pour un symbole S la valeur d'une propriété Pi

- **PUTPROP**: (PUTPROP symbole valeur propriété)
  -> met sous la propriété propriété du symbole symbole la valeur valeur.
  Rend la valeur.

- **PLIST**: (PLIST symbole)
  -> rend la liste de propriétés et de valeurs associé au symbole symbole.

- **PLIST**: (PLIST Symbole Liste)
  -> associe au symbole Symbole la Pliste Liste, et la rend

- **GETPROP**: (GETPROP symbole propriete)
  -> rend la valeur de la propriété propriete du symbole symbole.

- **REMPROP**: (REMPROP symbole propriete)
  -> enlève de la P-liste de symbole la propriété propriete et sa valeur associée. Rend la liste composée de cette propriété propriete et de sa valeur.

4. Extensions de la programmation objet

- D. Petit comparatif des LOO

- Référence faible (70)

- Réflexivité 75

- Référence sur une méthode (82)
- généricité 91

Historique

- 91 : SUN –> Echec
  - Pour de petits appareils (tel portable…): code embarqué
  - Machine virtuelle (Byte Code)
  - Syntaxe proche de C++

Bibliographie

- Claude Delannoy, programmer en JAVA, Eyrolles 2008, 5eme edition
- E. Puybaret, JAVA 1.4 et 5.0, Eyrolles 2004