Université Hassan 1er
Faculté des Sciences et Techniques
Département Mathématiques et Informatique

PROGRAMMATION ORIENTEE OBJET EN JAVA

Livre des travaux Pratiques

Projet: Library

Filière: Cycle d'Ingénieur

Génie Informatique

Professeur BALOUKI Youssef

Déparetement Mathématiques et Informatique

Table des matières

Question TP 1: Classes et Objets

Question de l'atelier Librairie: Objets et Classes

Dans cet atelier, vous allez:

- **Créer une classe.**
- **Créer un objet**
- **Déclarer un attribut.**
- **Surcharger un constructeur.**

Objectif

Créez la classe Document et ses constructeurs.

Etape 1: Création de la classe Document – Atelier individuel

Créez et compilez la classe *Document* contenant les attributs suivants:

Nom	Type	Valeur par défaut
Title	String	"X"
creationDate	Date	Date de création de l'objet
Pages	Int	-1

Ajoutez une méthode `toString()` qui retourne une chaine décrivant un Document.

```
public String toString()
```

Etape 2: Ajoutez le programme principal – Atelier individuel

Créez une classe Main.*java* contenant une méthode *main().* Celle-ci doit créer un *Document* et afficher la valeur de ses attributs par *toString()*.

Pour afficher un Document, écrivez:

```
System.out.println(doc.toString());
```

Ou seulement:

```
System.out.println(xx);
```

Vous devez obtenir le résultat suivant:

```
Title =X Pages =-1 Creation Date =Thu Jan 10 09:36:37 CET 2013
```

Etape 3: Surchage du constructeur – Atelier individuel

Les arguments de la ligne de commande doivent contenir: le titre et le nombre de pages permettant de construire un objet *Document*.

- Ecrivez un second constructeur prenant deux paramètres (titre et numéro de pages en utilisant le mécanisme de surchage.
- Modifiez le main() de manière à déclencher le bon construteur en fonction du nombre de paramètres de la ligne de commande.

En lançant le programme via

```
java Main "Programmation objet en Java"  450
```

vous devez obtenir:

```
Title =Programmation objet en Java Pages =450 Creation Date =Thu Jan 10 09:55:37 CET 2013
```

Et via

```
java Main
```

vous devez obtenir:

```
Title =X Pages =-1 Creation Date =Thu Jan 10 09:36:37 CET 2013
```

Remarques

Pour convertir une chaine de caractères en entier, utilisée la méthode *parseInt* de la classe *java.lang.Integer* (regardez dans la documentation du JDK).

Question TP2: Héritage

Question de l'atelier: Library-InheritancePackage

Vous allez pratiquer:

- **Créer des sous classes.**
- **Créer et utiliser des packages.**
- **Créer des méthodes et attributs.**

Objectifs

Une Librairie peut contenir des livres (Book) et des périodiques (Periodical):

- Un livre possède un nombre de pages, un titre, un auteur et un editeur
- Un périodique possède un nombre de pages, un titre, une fréquence de publication : MONTHLY, BIMONTHLY, QUATERLY.

Etape 0: Conception – Atelier en groupe

Définissez les packages du projet (dessinez un diagramme de package UML), et les classes (dessinez un diagramme de classes UML).

Packages

Class Diagram

Etape 1: Créez les packages et les classes – Atelier individuel

Dans le projet 02-Library-InheritancePackage :

Créez les packages library.bo et library.gui .

Déplacez la classe Document vers le package library.bo

Créez les classes Book et Periodical (vides) héritant de la classe Document dans le package approprié (library.bo). Les attributs et les méthodes seront ajoutés dans l'étape suivante

Créez la classe Library (vide aussi pour l'instant) au sein du même package.

Créer la classe MainStep1.java du package package library.gui de votre projet. Son main :

- Crée un objet de la classe Periodical et l'affiche via System.out.println(…)
- Crée un objet de la classe Book et l'affiche via System.out.println(…)
- Crée un objet de la classe Library et l'affiche via System.out.println(…)

Exécutez MainStep1, vous devez obtenir:

```
*** Inheritance-Package exercise - Step1 ***
library.bo.Periodical Title = X Pages = -1 Creation Date = Thu May 11 18:04:34 CEST 2006
library.bo.Book Title = X Pages = -1 Creation Date = Thu May 11 18:04:34 CEST 2006
library.bo.Library@7d772e
```

Etape 2: Ajouter les méthodes et attributs – Atelier individuel

Déplacez la classe MainStep2.java du package par défaut vers le package library.gui de votre projet.

Ajoutez les méthodes et attributs dans les classes *Library*, *Document*, *Book*, et *Periodical* de façon à faire fonctionner le main de MainStep2.java.

Exécutez MainStep2, vous devez obtenir:

```
The Valtech Library
library.bo.Book Title = NT 5 Pages = 600 Author = Bill Porte Editor = O'Reilly
library.bo.Book Title = JavaBeans Pages = 1200 Author = Vincent K. Bean Editor = McGriwHall
library.bo.Book Title = Internet Pages = 180 Author = Mark Cyberman Editor = Pocket
library.bo.Periodical Title = The Times Pages = 100 Freq = MONTHLY
library.bo.Periodical Title = Elle Pages = 40 Freq = BIMONTHLY
```

Suggestion : un enum est le meilleur moyen de représenter la fréquence de parution des Periodical

Question TP3: Interfaces

Question de l'atelier Library-Interface

Vous allez experiementer comment:

- **Créer une interface.**
- **Implémenter une interface.**

Objectif

L'association entre *Library* et *Document* est codée en utilisant un tableau. La méthode *documents()* retourne une référence sur le tableau: cette méthode brise l'encapsulation.

L'objectif de cet atelier est de résoudre ce problème, en utilisant une interface.

Etape 1: Conception – Atelier en groupe

Concevez une solution permettant au code suivant de fonctionner:
Projet 03-Library-Interface , package library.gui;

```
import library.bo.*;

public class Main {
    public static void main(String argv[]) {
        Library        lib = new Library("The Valtech Library");
        Book           book;
        Periodical     periodical;

        book = new Book("NT 5", 600, "Bill Porte", "O'Reilly");
        lib.addDocument(book);

[etc…]

        Scanner scanner = lib.documents();
        while( scanner.hasNext() ) {
          Document document = (Document)scanner.next();
               System.out.println(document.toString());
        }
    }
}
```

Ce code utilise une interface nommée *Scanner* permettant de parcourir les documents. Cette interface est générique et retourne un *Object*, non pas un *Document*.

Il faudra donc créer une Interface Scanner, et une classe fournissant son implémentation.

Ecrivez le pseudo code pour les méthodes *Scanner.hasNext()* et *Scanner.next().*

Etape 2: Implementation –Atelier Individuel

Ouvrez le projet 03-Library-Interface.

Pour faire fonctionner la classe library.gui.Main.java , implémentez votre conception.

Question TP4: Classes de base

Question de l'atelier Library-BaseClasses

Vous allez experimenter comment:

- **tester 2 objets**

Objectif

Vous devez tester si deux documents sont égaux:

- Deux livres (Book) sont égaux s'ils ont le même auteur et le même titre.
- Deux périodiques sont égaux s'ils ont le même titre et la même fréquence de parution.

Instructions

```
package library.gui;
import library.bo.*;
public class Main {
public static void main(String argv[]) {
     Book book1 = new Book("Enterprise JavaBeans", 600, "Bill Porte", "O'Reilly");
     Book book2 = new Book("Enterprise JavaBeans", 600, "Bill Porte", "O'Reilly");
     Book book3 = new Book("Enterprise JavaBeans", 600, "Billou", "O'Reilly");

     Periodical per1 = new Periodical("Elle", 40, Periodical.frequency.BIMONTHLY);
     Periodical per2 = new Periodical("Elle", 40, Periodical.frequency.BIMONTHLY);
     Periodical per3 = new Periodical("Elle", 40, Periodical.frequency.MONTHLY);

     if( book1.equals(book2) )
        System.out.println("Test 1 Book is OK");
     else
        System.out.println("ERROR: test 1 Book");

     if( book1.equals(book3) )
        System.out.println("ERROR: test 2 Book");
     else
        System.out.println("Test 2 Book is OK");

     if( per1.equals(per2) )
        System.out.println("Test 1 Periodical is OK");
     else
        System.out.println("ERROR: test 1 Periodical");

     if( per1.equals(per3) )
        System.out.println("ERROR: test 2 Periodical");
     else
        System.out.println("Test 2 Periodical is OK");

     if( book1.equals(per1) )
        System.out.println("ERROR: last test");
     else
        System.out.println("Last Test is OK");

  }}
```

1. Modifier le code pour que le main de *library.gui.Main* fonctionne :

```
*** Base classes exercise ***
Test 1 Book is OK
Test 2 Book is OK
Test 1 Periodical is OK
Test 2 Periodical is OK
Last Test is OK
```

Conseil : redéfinissez la méthode *boolean equals(Object)* héritée de la classe Object pour les classes concernées (Document, Book & Periodical).

Question TP5: Les collections

Question de l'atelier: Library-Collection

Vous allez:

- **Utiliser les collections java2.**
- **Utiliser la HashMap**

Objectif

Vous devez remplacer le tableau stockant les documents de la bibliothèque par une collection (utilisez une collection générique Java 5 !).

Ensuite, vous ajouterez des indexes permettant de récupérer les documents à partir de mots clefs.

Etape 1: Remplacez le tableau par une collection – Atelier individuel

Modifiez la classe Library afin de faire fonctionner le main de la classe library.gui.MainStep1 :

- Remplacez le tableau de documents par une collection
  (pensez à manipuler la collection à travers une interface)
- La méthode *documents*() doit retourner un *Iterator<Document>*.
- La méthode *addDocument()* ajoute un document dans la bibliothèque.

Etape 2: Ajoutez des indexes – conception : Atelier de groupe

Un document peut être référencé par un ou plusieurs indexes. Un index peut référencer un ou plusieurs documents.

Ajoutez les méthodes suivantes :

- *void addIndex(Document doc, String index)*
  Qui indique que le document doit être référencé par l'index spécifié
  (si le document n'est pas encore dans la librairie, il sera ajouté)
- Iterator<String> *indexes()* .
  Qui retourne un Iterator sur une collection (en pratique un Set) des index de la librairie
- Iterator<Document> *findOutDocumentsFromIndex(String index)*
  Qui retourne un itérateur sur la liste des documents referencés par l'index spécifié
- Iterator<String> *findOutIndexesFromDocument(Document doc)*
  Qui retourne un itérateur sur la liste des index référençant le document spécifié

Etape 2: Ajoutez des indexes – implémentation : Atelier individuel

Ajout des méthodes spécifiées ci-dessus :

1. Ajoutez une *HashMap* et les méthodes *addIndex()* et *indexes() .*
   Testez en faisant fonctionner la classe *library.gui.MainStep23(partie 1).*
2. Ajoutez la méthode *findOutDocumentFromIndex() .*
   Testez faisant fonctionner la classe *library.gui.MainStep23(partie 2)*
3. Ajoutez la méthode *findOutIndexesFromDocument().*
   Testez en faisant fonctionner la classe *library.gui.MainStep23(partie 3)*

Note: Ne changez pas la classe Document. Seule la classe Library doit être modifiée.

```
package library.gui;

import library.bo.*;

import java.util.*;
import java.io.*;

public class MainStep23 {
  public static void main(String argv[]) {
    Library         library = new Library("Valtech Library");
    Book            book;
    Periodical      periodical;

    // Creation d'un ensemble de Documents
    book = new Book("NT 5", 600, "Bill Porte", "AW");
    library.addDocument(book);
    library.addIndex(book, "Computers");
    library.addIndex(book, "Linux Rules");
    library.addIndex(book, "Kernel Error");

    book = new Book("JavaBeans", 1200, "Vincent Legrain", "McGH");
    library.addDocument(book);
    library.addIndex(book, "Computers");
    library.addIndex(book, "Java");

    book = new Book("Internet", 180, "Mark Cyberman", "Pocket");
    library.addDocument(book);
    library.addIndex(book, "Internet");
    library.addIndex(book, "Computers");

    periodical = new Periodical("Times", 100,
    Periodical.Frequency.MONTHLY);
    library.addDocument(periodical);
    library.addIndex(periodical, "Journey");
    library.addIndex(periodical, "Quebec");

    periodical = new Periodical("Elle", 40,
    Periodical.Frequency.BIMONTHLY);
    library.addDocument(periodical);

    library.addIndex(periodical, "Computers");
    library.addIndex(periodical, "Java");

 // Lister les Documents
    System.out.println(library.getName());

    for( Iterator<Document> documents = library.documents();
         documents.hasNext(); )
         System.out.println(documents.next());
    pause();
```

```
// Afficher les Indexes
    System.out.println("\n Liste des  Indexes");
    for( Iterator<String> indexes = library.indexes();
         indexes.hasNext(); )
         System.out.println("  " + indexes.next());
         pause();


 System.out.println("\nDocuments par indexe");
 for( Iterator<String> indexes = library.indexes();indexes.hasNext(); ) {
         String index = indexes.next();
         System.out.println("Index : " + index);

    for(  Iterator<Document> documents = library.findOutDocumentsFromIndex(index);
              documents.hasNext(); )
              System.out.println("  " + documents.next());
    }

    pause();

    System.out.println("\nIndexe par document");
for(    Iterator<Document> documents = library.documents();documents.hasNext(); ) {
            Document document = documents.next();
            System.out.println("Document : " + document);

       for( Iterator<String> indexes = library.findOutIndexesFromDocument(document);
                 indexes.hasNext(); )
                 System.out.println("  " + indexes.next());
    }
  }

private static BufferedReader stdin = new BufferedReader(
                                                 new InputStreamReader(System.in));

  static void pause() {
    try {
      System.out.println("Enter to continue");
      String line = stdin.readLine();
    } catch(Exception e) {}
  }
}
```

Question TP6: Les exceptions

Question de l'atelier Java Exception

Vous allez...

- **Créer une exception**
- **Traiter une exception**

Objectif

Définissez l'exception *NotFoundException* qui est levée par les méthodes *findOutDocumentFromIndex()* et *findOutIndexesFromDocument()* .

Instructions

Faites fonctionnner le main de la classe *library.gui.Main*.

Note : ne modifiez que la classe *library.bo.Library*.

Question TP7: Swing

Question de l'atelier Library-Swing

Vous allez...

- **Assembler des composants**
- **Utiliser des LayoutManager**
- **Traiter les événements**
- **Afficher une boite de dialogue**

Objectif

Construisez une interface utilisateur permettant d'afficher les documents d'une bibliothèque.

Cet atelier est découpé en 4 étapes:

- Construire la fenêtre principale,
- Traiter les événements,
- Afficher les documents et les indexes,
- Editer les documents avec une boite de dialogue (étape optionnelle).

Conseil: concevez la solution avant de la coder !

Conseil: conservez un source bien lisible.

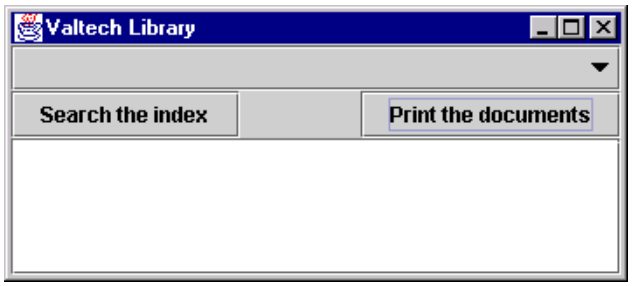
Etape 1: Assemblez les composants – Atelier individuel

Ouvrez le projet 07-Library-SwingFdm

Créez la fenêtre principale de l'application en assemblant des composants Swing à partir de la classe library.gui.FrameLibrary qui sous classe JFrame.

N'utilisez pas un outil d'assemblage java; vous devez taper le code.

Vous devez obtenir:



Etape 2: Conception – Atelier en groupe

Concevez une solution pour que les différents boutons de l'interface utilisateur fonctionnent.

- « Print the documents » affiche les documents dans la liste.

- « Search on Index » : affiche les documents correspondants à l'index sélectionné.
- La ComboBox affiche la liste des indexes de la bibliothèque.
- L'application doit se terminer sur l'utilisateur clique sur la croix.

Modifiez votre modèle, créez un diagramme de séquence pour « print the document »

Etape 3 : Implémentez la conception, première partie – Atelier individuel

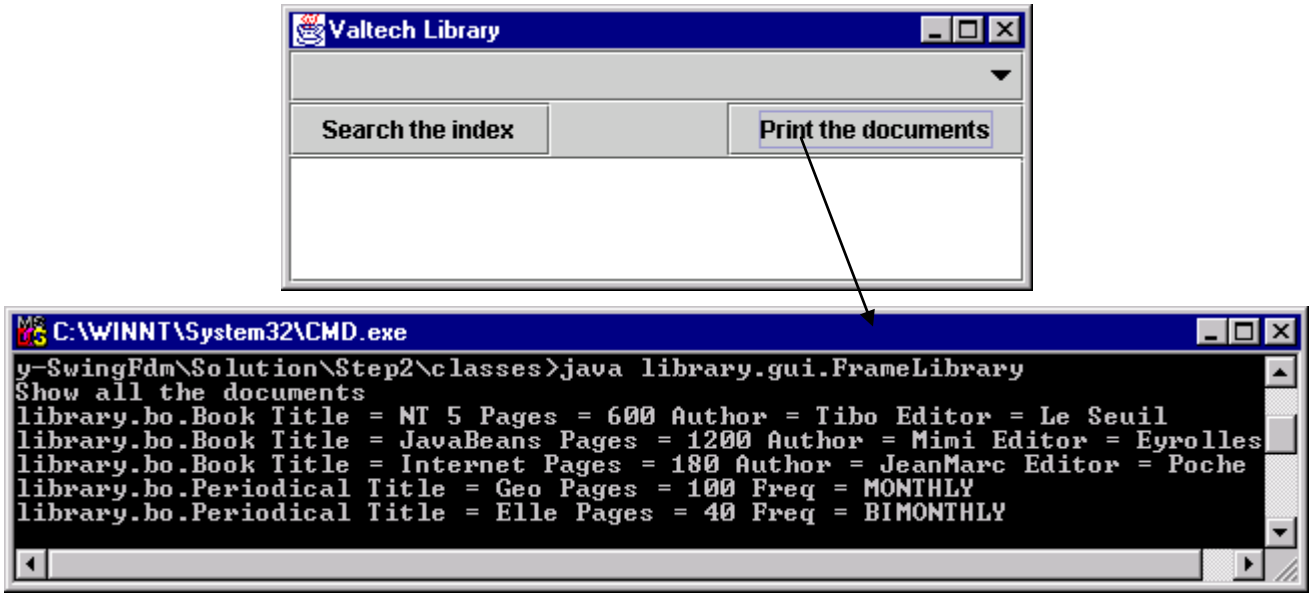
Si vous cliquez sur "Print the documents", vous devez afficher les documents de la bibliothèque dans la console Java par System.out.println().

Vous devez aussi traiter l'événement permettant de fermer la fenêtre si l'utilisateur clique sur la croix en haut à droite. 

Pour sortir d'une application Java:

```
JFrame.setVisible(false);
JFrame.dispose();
System.exit(1);
```

Vous devez obtenir:



Etape 4: Implémentez la conception, deuxième partie – Atelier individuel

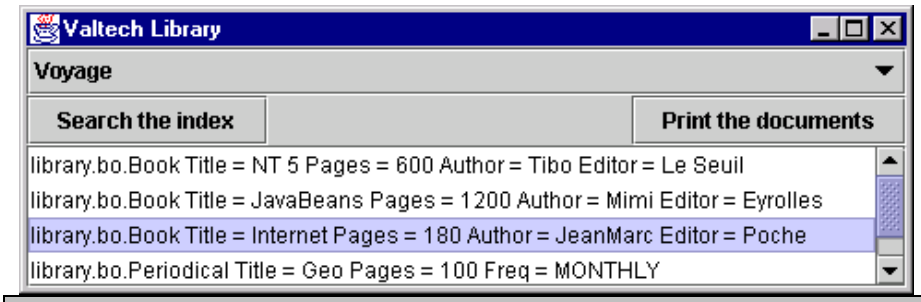
1- Affichez les documents dans une JList lorsque l'utilisateur clique sur "Print the documents".

2- Affichez la liste des indexes dans la JComboBox.

3- Faite fonctionner le bouton "Search on index" qui permet d'afficher dans la JList les documents correspondant à l'index sélectionner.

Procédez étape par étape.

Vous devez obtenir:



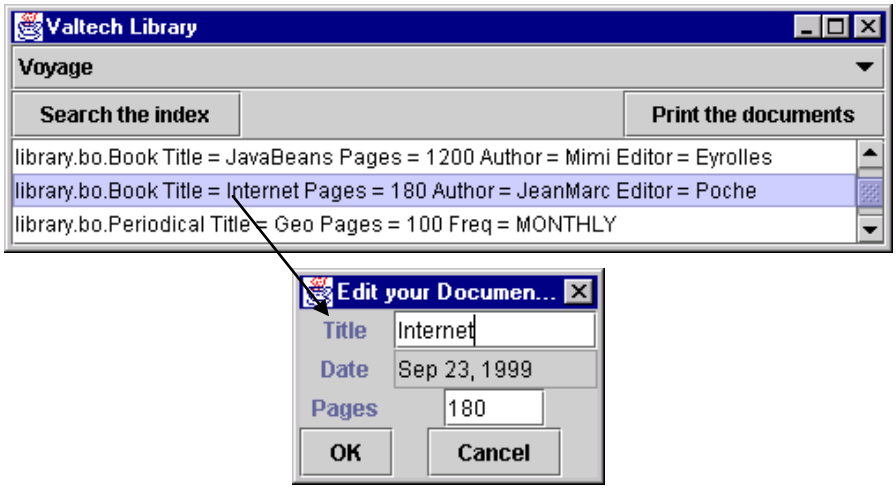
Etape 5: Ajoutez une boite de dialogue (optionnel)

Si l'utilisateur double clique sur un document de la liste, affichez une boite de dialogue permettant de modifier le titre et le nombre de pages.

Pour gérer le double clique sur une liste, voyez la documentation du JDK de la JList.

La boite de dialogue doit sous-classer JDialog et contenir des JLabel et JTextField.

Vous devez obtenir:



Question TP8: Java IO

Question de l'atelier: Java IO

Vous allez apprendre à:

- **Sérialiser un graphe d'objet.**
- **Ouvrir et fermer un fichier**

Objectifs

Au sein du projet 08-Library-IO, ajoutez un bouton *Open* et *Save* dans votre application.

- *Save* permet de sérialiser la librairie dans un fichier.
- *Open* charge les documents à partir d'un fichier.

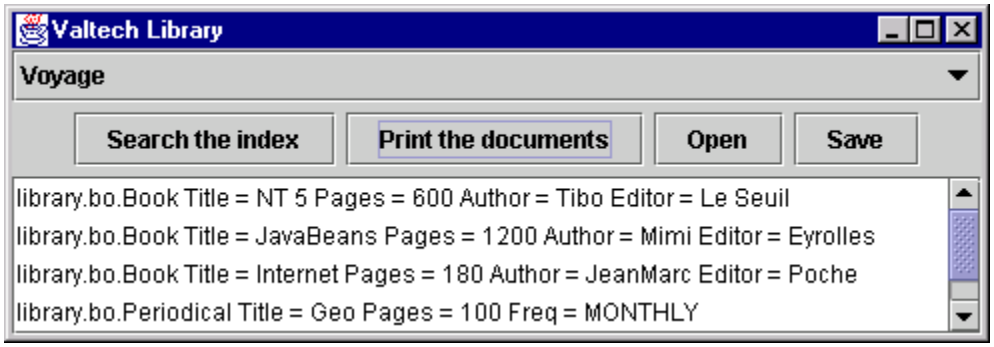
Etape 1 : Conception – Atelier de groupe

Modifiez votre modèle objet pour concevoir une solution.

Tracez un diagramme de séquence pour définir les nouvelles responsabilités (méthodes) des classes.

Etape 2 : Implémentation – Atelier individuel

Vous devez obtenir:



Notes :

- la classe à sérialiser est la classe Library (et ses membres)
- elle devra proposer des méthodes save() et open(). Note : open() devra être accessible sans qu'une instance de la librairie ait été créée (sta…)
- Chargement de la librairie : il sera nécessaire de réinitialiser le modèle de la ComboBox affichant les indexes, une fois la librairie chargée (sinon, aucun index ne s'affichera).

Question TP9: JDBC

Questiond de l'atelier Library-JDBC

Vous allez apprendre à:

- **Ouvrir une connexion JDBC.**
- **Exécuter une requête SQL.**
- **Traiter le résultat d'une requête SQL.**

Objectif

Les documents doivent être stockés dans une base de données : vous devez donc ajouter un connecteur JDBC permettant de lire et de modifier les données.

Pour cet atelier, vous allez utiliser la base de donnée HSQLdB se trouvant dans le répertoire Library-JDBC/database.

Configuration

- Lancer le serveur de base de données.
- Pour cela, aller dans le répertoire c:\stagiaire\database
- Lancer le programme runServer

Les paramètres de connexions à la base sont les suivants :

Type = HSQL Database Engine Server
Driver = org.hsqldb.jdbcDriver
URL = jdbc:hsqldb:hsql://localhost/
User = sa
Pasword = "" (rien, pas de mot de passe)

Vous pouvez tester que la base est correctement lancée, en utilisant le script *runManager.*

Etape 1: Conception – Atelier de groupe optionnel.

Pour cet atelier, vous pouvez travailler sur la conception ou le formateur vous présente directement la solution.

Concevez une solution permettant de rendre vos objets Java persistants dans la base de données.

Définissez le modèle de stockage sous la forme d'un schéma relationnel ainsi que le mapping entre le modèle objet Java et le modèle de stockage.

Ensuite, définissez les classes permettant de gérer la persistance des objets. Pour simplifiez, ne traitez que l'accès aux données (SELECT), pas les modifications (UPDATE, DELETE, CREATE) .

Votre conception doit :

- Etre la plus simple possible
- Le code SQL ne doit pas être inclus dans les objets métier (Document, Book…)
- La performance n'est pas très importante pour cet atelier

Etape 2: Affichez la liste des indexes (Topics) – Atelier individuel.

Ouvrez le projet 09-Library-JDBC.

Remplissez la comboBox avec la liste des indexes venant de la base de données.

Vous devez modifier le code du projet.

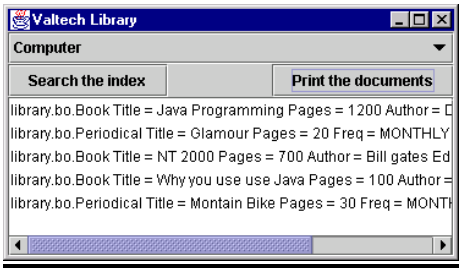
- Implémentez la méthode Iterator<String> *TopicDAO.indexes()*

La base de données fournie contient deux tables : TOPIC (index) et DOCUMENT. Seule la table TOPIC sera utilisée pour cette étape.

- La table TOPIC contient les colonnes suivantes:
  - TOPIC (PK, String): l'index
  - DOCUMENT (PK, FK, String): le titre du document
- Modifiez les classes:
  - Database
  - TopicDAO
- La requête SQL a exécuter est:
  - SELECT DISTINCT topic FROM topic

Etape 3: Afficher tous les documents – Atelier individuel.

Le bouton « print the documents » soit retrouver les données à partir de la base.

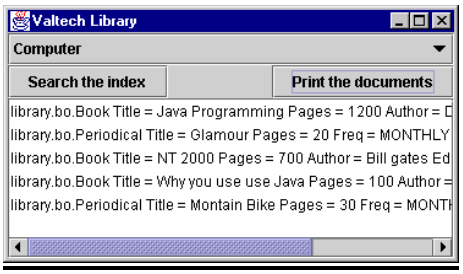


- La table DOCUMENT contient les valeurs suivantes :
  - **CLASS (int): 1 pour Book, 2 pour Periodical**
  - **TITLE (PK, String): titre du document**
  - **PAGES (int): nombre de pages**
  - **AUTHOR (String): Auteur, que pour un Book**
  - **EDITOR (String): Editeur, que pour un Book**
  - **FREQUENCY (String): Frequence, que pour un Periodical: MONTHLY, BIMONTHLY, QUARTERLY**
- La table *DOCUMENT* n'est pas normalisée. La classe *DocumentDAO* doit créer le bon objet (*Book* or *Periodical*) en fonction de la valeur de la colonne *CLASS*.

Note: Cette étape vous montre comment un modèle logique peut isoler une application du modèle physique de stockage.

- Implémentez la méthode *Library.documents()*;

Etape 4: Affichez les documents en fonction d'un index – Atelier individuel optionnel



Le bouton « Search on index » doit récupérer les documents correspondant à un index à partir de la base de données.

- Implémentez la méthode Library.findOutDocumentsFromIndex(String).
- La requête SQL est (pour l'index « Java »):
  - SELECT Document.* FROM Document, Topic WHERE Document.TITLE = Topic.DOCUMENT AND Topic.TOPIC='Java';