Java Avancé - Cours 1

Plan

La version électronique de ce cours est disponible à l'URL `http://www.derepas.com/java`.

Un facteur clé de succès pour un langage actuel est de pouvoir réutiliser des librairies existantes, de pouvoir communiquer avec des bases de données, ou d'autres programmes en train de s'exécuter. Ce cours met en avant la communication sur internet à l'aide de Java.

1 Recevoir et émettre des données simples

Le but de cette partie est d'écrire un petit programme java qui va chercher une page donnée sur le Web.

1.1 Comment récupérer une page Web

Pour récupérer la page correspondant à l'adresse `http://www.yahoo.com`, il faut suivant le protocole http:

- se connecter sur le port 80 de la machine www.yahoo.com
- envoyer la suite de caractères:

```
GET / HTTP/1.1
Host: www.yahoo.com:80
```

suivit d'une ligne vide.

1.2 Socket : l'objet qui permet de manipuler une connexion

L'objet Java qui permet de manipuler une connexion est `java.net.Socket`. Pour ouvrir une connexion sur le port 80 de la machine `www.yahoo.com`, il faut initialiser la socket avec la séquence :

```
Socket socket = new Socket ("www.yahoo.com", 80);
```

Pour envoyer des caractères on peut récupérer un objet de type `java.io.PrintWriter`:

```
PrintWriter out = new PrintWriter(socket.getOutputStream(), true);
out.println("Chaine a envoyer");
```

1.3 Que faire en cas d'erreur ?

Quand on communique avec une autre machine beaucoup d'incidents peuvent se produire : la machine distante peut être éteinte à ce moment, le réseau peut être saturé, la carte réseau peut tomber en panne.

Dans la documentation Java on remarque que le prototype du constructeur Socket est:

```
public Socket(InetAddress address, int port) throws java.io.IOException
```

la clause `throws IOException` signifie que la séquence d'instruction:

```
Socket socket = new Socket ("www.yahoo.com", 80);
```

peut créer une anomalie (exception en anglais) qui sera une instance de la class `java.io.IOException`.

1.4 Exemple d'exception

Les exception en Java ont le sens suivant:

> Essaye la commande `new Socket ...` en cas d'erreur de type `java.io.IOException` alors fait telle ou telle action.

La syntaxe utilisée est la suivante:

```
try {
    Socket socket = new Socket ("www.yahoo.com", 80);
} catch (java.io.IOException e) {
    // une erreur c'est produite, le signaler !
    System.err.println("erreur de type I/O");
}
```

1.5 Bien exploiter les exceptions

Quand une exception de type `java.io.IOException` est récupérée par une instruction `catch` on dispose de plusieurs informations qui peuvent être exploitées, principalement une description de l'erreur obtenue en invoquant la méthode `getMessage` et l'endroit ou l'erreur s'est produite en invoquant `printStackTrace`.

Considérons le programme suivant:

```
class ServerError {
    public static void essayeLaConnexion() {
        try {
            java.net.Socket socket =
                new java.net.Socket ("www.serveurquinexistepas.com", 80);
        } catch (java.io.IOException e) {
            // une erreur c'est produite, le signaler !
            System.err.println(e.getMessage());
            e.printStackTrace();
        }
    }
    public static void main(String[] args) {
        essayeLaConnexion();
    }
}
```

Son exécution donne l'affichage suivant:

```
www.serveurquinexistepas.com
java.net.UnknownHostException: www.serveurquinexistepas.com
        at java.net.PlainSocketImpl.connect(PlainSocketImpl.java:153)
        at java.net.Socket.connect(Socket.java:452)
        at java.net.Socket.connect(Socket.java:402)
        at java.net.Socket.<init>(Socket.java:309)
        at java.net.Socket.<init>(Socket.java:124)
        at ServerError.essayeLaConnexion(ServerError.java:4)
        at ServerError.main(ServerError.java:13)
```

Ceci nous permet tout de suite de savoir que l'erreur s'est produite à la ligne 4 dans la méthode `essayeLaConnexion` qui était appelée par `main` à la ligne 13.

1.6 Hiérarchie des exceptions

Les exception étant des objets Java comme les autres, ils ont donc leur propre hiérarchie d'héritage. En effet dans l'exemple précédent l'erreur levée n'est pas `java.io.IOException` mais: `java.net.UnknownHostException` nous indiquant que le serveur `www.serveurquinexistepas.com` n'existe pas.

La documentation Java nous donne la hiérarchie suivante:

```
java.lang.Object
  java.lang.Throwable
      java.lang.Exception
          java.io.IOException
              java.net.UnknownHostException
```

Ainsi le sens de la commande `catch(IOException e)` est : rattrape toute exception de type `IOException` ou d'un type hérité.

2 JDBC

2.1 Présentation

Au sein du système d'information d'une entreprise enormément de données sont stokées. Elles le sont généralement sous forme de base de données relationnelles.

Il existe également des bases de données objet, c'est à dire où l'on peut directement charger des objets java de la base vers l'application. Ce type base se développe mais reste encore marginal au sein des systèmes d'informations.

JDBC est l'interface java pour accéder aux bases de données relationnelles. Chaque base doit proposer son driver JDBC qui permet d'y accéder. Quelque soit la base, le driver JDBC se manipule de la même manière.

2.2 Se connecter à la base de données

Nous prendrons l'exemple de mysql `www.mysql.com` une base de données open source répandue. Les drivers JDBC sont téléchargeable à l'url : `http://dev.mysql.com/downloads/connector/j`.

Pour se connecter à la base il faut tout d'abord charger le driver fourni par Mysql.

Ceci s'effectue par le code suivant:

```
Class.forName("com.mysql.jdbc.Driver");
```

Pour obtenir une connexion sur la base il faut alors utiliser la class `java.sql.DriverManager`:

```
java.sql.Connection conn =
  java.sql.DriverManager.getConnection
    ("jdbc:mysql://localhost/nomdemabase?user=monlogin&password=motdepasse");
```

La chaîne de caractères après `jdbc:`... est généralement assez dépendante de la base de donnée, et n'est valable ici que pour mysql. Ici on se connecte sur la machine locale (`localhost`), à la base nommée `nomdemabase` sous le nom `monlogin` avec le mot de passe `motdepasse` écrit en clair dans le fichier.

2.3 Envoyer une requête

Une fois la connexion établie on peut envoyer une requête à la base comme indiqué ci-dessous :

```
java.sql.Statement stmt = conn.createStatement();
java.sql.ResultSet rs = stmt.executeQuery("SELECT host, user FROM user");
```

Ici la requête SQL effectuée est `SELECT host, user FROM user`. Bien entendu des exceptions SQL sont levées si la requête est incohérente.

Pour les mises à jour dans une table on utilise la méthode `executeUpdate` ne renvoyant rien à la place de `executeQuery`. Ce qui donne:

```
String updateString = "UPDATE matable " +
      "SET monchamp = 32 " +
      "WHERE id='43'";
stmt.executeUpdate(updateString);
```

2.4 Afficher les résultats

Pour afficher les résultats d'une requête il faut interroger l'objet `ResultSet` comme suit :

```
while (rs.next()) {
    String h = rs.getString("host");  // recupere la valeur du champ host
    String u = rs.getString("user");  // recupere la valeur du champ user
    System.out.println(h+ "   " + u); // affiche le resultat
}
```

2.5 Libérer les ressources

Les actions précédents ont conduit à une communication avec la base et la réservation de ressources dans cette dernière. Il est bon de libérer ces ressources en appelant les méthodes suivantes sur les objets créées :

```
rs.close();
stmt.close();
```

Cependant si une exception survient, la procédure en cours est interrompue. Si l'on souhaite relacher quand même les ressources, il faut utiliser la séquence suivante:

```
try {
  // code qui declenche une exception
  ...
} catch ( ... ) {
  // actions en cas d'erreurs
  ...
} finally {
  // actions a faire toujours en quittant le block try-catch
  ...
}
```

Ainsi dans le block finally on va trouver: `rs.close()` et `stmt.close()`. Cependant l'appel à la méthode `close` pouvant lever des exceptions, il faut elle même la mettre dans un block try-catch. La séquence complète est donc :

```
try {
    ...
} catch ( ... ) {
    ...
} finally {
    // relacher les ressources dans l'ordre contraire d'allocation
    if (rs != null) {
        try {
            rs.close();
```

```
                } catch (SQLException sqlEx) {
                    // ignorer
                }
                rs = null;
            }
            if (stmt != null) {
                try {
                    stmt.close();
                } catch (SQLException sqlEx) {
                    // ignorer
                }
                stmt = null;
            }
        }
```

2.6 Requêtes répétitives

Dans le cas de requêtes répétitives, il existe un objet spécial qui permet de gagner en temps d'exécution : `java.sql.PreparedStatement`.

Par exemple on souhaite effectuer de manière répétée les insertions suivantes:

```
UPDATE matable SET monchamp = ? WHERE id=?
```

où les `?` sont à remplacer par plusieurs couples de valeurs.

```
PreparedStatement miseAjourTable = conn.prepareStatement(
    "UPDATE matable SET monchamp = ? WHERE id=?");
```

Alors pour effectuer la requête SQL `UPDATE matable SET monchamp = 'toto' WHERE id=12`, on peut écrire:

```
miseAjourTable.setString(1, "toto");
miseAjourTable.setInt(2, 12);
miseAjourTable.executeUpdate():
```

Pour plus d'informations vous pouvez consulter l'url :

`http://java.sun.com/docs/books/tutorial/jdbc/index.html`

3 XML

3.1 But

Les fichiers XML sont très souvent présents. Ils peuvent permettre d'échanger des données entre des applications, ou encore de garder des données de configuration.

3.2 Rappel sur XML

Voici un rapide aperçu d'XML. Les fichiers Xml sont des fichiers ASCII commençant par (ici pour de un encodage UTF-8):

```
<?xml version="1.0" encoding="UTF-8"?>
```

On trouve ensuite une balise, nommée racine, commençant le document (par exemple ici `fichierdeconf` est le nom de la balise) :

```
<fichierdeconf id="1">
```

et la balise symétrique la la fin du document :

```
</fichierdeconf>
```

On remarque la présence de l'attribut `id="1"`.

Entre `<fichierdeconf>` et `</fichierdeconf>` on trouve toute une série de balises. Toute balise `<mabalise>` doit être fermée par un balise `</mabalise>` correspondante. La balise d'entrée peut avoir des attributs:

```
<mabalise attribut1="valeur 1" id="3" toto="tata">
  <autrebalise> ... </autrebalise>
</mabalise>
```

Enfin on autorise l'abbréviation `<mabalise/>` pour `<mabalise></mabalise>`.

Voici le fichier Xml complet :

```
<?xml version="1.0" encoding="UTF-8"?>
<fichierdeconf id="1">
  <mabalise attribut1="valeur 1" id="3" toto="tata">
    <autrebalise> texte </autrebalise>
  </mabalise>
</fichierdeconf>
```

3.3 Obtenir la racine du document

On crée tout d'abord une entitée nomée `DocumentBuilder` qui va lire le fichier :

```
javax.xml.parsers.DocumentBuilderFactory factory =
    javax.xml.parsers.DocumentBuilderFactory.newInstance();
javax.xml.parsers.DocumentBuilder builder = factory.newDocumentBuilder();
org.w3c.dom.Document document = builder.parse( new java.io.File("fichier.xml"));
```

La racine du document s'obtient alors par :

```
org.w3c.dom.NodeList rootChild = document.getChildNodes();
org.w3c.dom.Node racine = rootChild.item(0);
```

3.4 Informations sur un noeud

Considérons le nœud `racine` de type `org.w3c.dom.Node` sur le fichier XML donné en exemple à la section 3.2.

Le nom de la balise (`fichierdeconf`) est alors obtenu par `racine.getNodeName()` . La valeur de l'attribut `id` du noeud `racine` est obtenue par :

```
org.w3c.dom.NamedNodeMap m = racine.getAttributes();
String valeurDeId = m.getNamedItem("id").getNodeValue()
```

Ou si l'on sait que la valeur est un entier :

```
int i = Integer.parseInt(racine.getNamedItem("id").getNodeValue());
```

L'instruction précédente provoque une exception si aucun attribute `id` est présent dans le noeud `n`.

3.5 Informations sur la descendance d'un noeud

Pour faire une action sur tous les enfants de `n` de type `org.w3c.dom.Node`, il faut utiliser la méthode `getChildNodes` :

```
org.w3c.dom.NodeList children = racine.getChildNodes();
if ( children != null ) {
    len = children.getLength();
    for ( int i = 0; i < len; i++ ) {
        org.w3c.dom.Node n = (org.w3c.dom.Node)children.item(i);
        System.out.println("  sous-noeud : "+n.getNodeName());
    }
}
```

Avec l'exemple précédent le résultat affiché est :

```
  sous-noeud : #text
  sous-noeud : mabalise
  sous-noeud : #text
```

L’exemple complet est téléchargeable aux urls :

```
http://www.derepas.com/java/XmlTest.java
http://www.derepas.com/java/toto.xml
```

Pour plus d’informations sur la lecture des fichiers XML en java :

```
https://jaxp.dev.java.net
```

4 Exercices

Exercice 1.1

Le but de l'exercice est de réaliser un programme affichant le code HTML d'une URL donnée.
Q1 Dans la section 1.2 on génère un objet de la classe `java.io.PrintWriter`, pour envoyer des données. De manière similaire, construire un objet o_1 de type `java.io.InputStreamReader` à partir de la socket, puis un objet o_2 de type `java.io.BufferedReader` pour lire les données recues par la socket.
Q2 Quelles sont les différences entre les classes `java.io.BufferedReader` et `java.io.InputStreamReader` ?
Q3 Ecrire un programme qui affiche la page `etudiants.masters.epita.net`.
Q4 En utilisant la classe `java.net.URL` écrire un programme qui affiche le contenu d'une URL qui lui est donnée en argument.

Exercice 1.2

Q1 Utiliser la classe `java.net.ServerSocket` pour écrire un serveur sur le port 8888 qui envoie le message `bonjour!` à tout client se connectant dessus. Tester le serveur avec le client écrit dans l'exercice précédent.
Q2 Se servir du serveur de la question 1 pour regarder quels sont les caractères envoyés par un navigateur comme internet explorer ou Netscape.

Exercice 1.3

Le but de cet exercice est de réaliser un filtre qui lit un fichier XML et l'écrit dans une base SQL.

On considère la base de données `ex3` crée par la suite de commandes mysql suivantes:

```
> create database ex3;
> use ex3;
> create table personnes (
      primary key (id),
      id int NOT NULL AUTO_INCREMENT,
      login text,
      nom text,
      prenom text);
>insert into personnes values (0,'ademusset','De Musset','Alfred');
>insert into personnes values (0,'gsand','Sand','Georges');
```

Q1 Écrire une classe `Personne` ayant les champs `id`, `login`, `nom` et `prenom` et un constructeur ayant en argument l'identifiant, et qui initialise les valeurs des champs en lisant la base de données.

Indication : la requête SQL pour obtenir la ligne d'identifiant 3 est `select login, nom, prenom from personnes where id=3;`
Q2 Dans la classe `Personne` ajouter un constructeur qui prend en argument un noeud XML (de type `org.w3c.dom.Node`) et construit l'instance correspondant à une séquence Xml de la forme:

```
<personne login="fchopin" prenom="frederic" nom="chopin"/>
```

Q3 Ecrire une classe qui prend en entrée un nom de fichier XML contenant une séquence de personnes et ajoute ces personnes dans la base. Le format du fichier Xml pourra être le suivant:

```
<?xml version="1.0" encoding="UTF-8"?>
<liste>
  <personne login="fchopin" prenom="Frederic" nom="Chopin"/>
  <personne login="gbizet" prenom="Georges" nom="Bizet"/>
</liste>
```

Indication : la requête SQL pour renter une nouvelle ligne est: `insert into personnes values(0, 'gbizet', 'Bizet', 'Georges')`.