Sujet de TP 2
Framework JEE
Message-Driven Beans

Chouki Tibermacine - chouki.tibermacine@umontpellier.fr

Objectifs du TP

- Programmer un composant MessageDriven Bean (MDB)
- Utiliser le composant MDB avec d'autres composants (Web et session stateful)
- Déployer une application avec un MDB dans un serveur d'application

Exercice

- L'objectif de cet exercice est d'étendre l'application écrite dans le premier sujet de TP sur les EJB, pour que la conversion soit effectuée dans toutes les monnaies disponibles, et que les résultats soient transmis par courrier électronique à l'utilisateur s'il le souhaite. Dans ce dernier cas, il doit indiquer sur l'interface Web une adresse email.
- Modifier le composant Web pour permettre à l'utilisateur d'indiquer en plus du montant à convertir, et de la monnaie cible, son adresse email
- Lorsqu'une adresse email est indiquée, le composant Web enverra un message sur une file de messages (Queue), écoutée par un composant MDB. Ce dernier va solliciter le bean session pour effectuer les conversions
- Modifier le bean session pour convertir le montant dans toutes les monnaies disponibles (le bean session devra donc fournir deux méthodes)
- Écrire le composant MDB qui récupère le message dès sa disponibilité dans la file, invoque la méthode du bean session, puis envoie un email avec les résultats de conversion
- Déployer toute l'application dans le serveur d'application WildFly et la tester.

Procédure à suivre.

- Démarrer votre IDE pour créer un nouveau serveur d'applications qui prend en compte les MDB (le serveur configuré par standalone.xml n'est pas adapté)
  - Dans le menu, choisir Edit Configurations...
  - Cliquer sur le bouton + pour créer une nouvelle configuration

— Choisir JBoss Local puis indiquer le chemin vers WildFly. Donner un nom au serveur (WildFly Full, par ex.)
— Dans l'onglet "Startup/Connection", vous avec un champ "Startup Script", décoher la case "Use default" et ajouter à la fin de la ligne (après standalone.bat) : `-c standalone-full.xml`
— Valider sur OK
— Cliquer avec le bouton droit sur le nouveau serveur WildFly dans la tool-window Services, puis choisir "Run"

▶ Modifier le composant Web de l'application (Converter) du TP précédent :
— Modifier la page JSP (index.jsp) pour ajouter au formulaire la possibilité de saisir une adresse email. Lors de la soumission de ce formulaire, c'est (toujours) ce même script qui va s'exécuter
— Mettre en place le code suivant :

```
String email = request.getParameter("email");
if(email != null && email.length() != 0) {
  // Demander au MDB de declencher la demande de conversion
  // dans toutes les monnaies par le bean session
  // le MDB va ensuite envoyer un email avec toutes
  // ces informations en HTML (voir plus loin ce que doivent
  // faire les beans) ...
}
```

— Ajouter au bean session deux méthodes métier (getAvailableCurrencies() et euroToOtherCurrencies(double amount)) qui retournent respectivement une `List` d'objets `Monnaie` pour la première et un `Map<Monnaie,Double>` pour la seconde, avec comme « clés » les monnaies et comme « valeurs » les résultats de conversion du montant dans chaque monnaie. On représentera les monnaies par des objets créés à partir d'une classe (Monnaie) qui comporte quatre attributs : le nom du (ou des) pays (une List de String), le nom complet de la monnaie (String), le code de la monnaie (String), et le taux de change actuel (float). Cette classe doit implémenter l'interface java.io.Serializable pour que ses objets puissent être sérialisés et transmis ou retournés dans des invocations de méthodes distantes (*Java RMI*). Pour obtenir le taux de change, nous allons utiliser le même document XML que dans le TP précédent, et pour obtenir les noms des pays correspondants à un code de monnaie et les noms complets des monnaies, nous allons analyser le document XML qui se trouve sur le site officiel ISO suivant : `https://www.currency-iso.org/dam/downloads/lists/list_one.xml`
Remplir d'abord la liste des monnaies à partir du premier document XML (celui de la banque centrale européenne, qui comporte peu de monnaies), et compléter cette liste avec les noms complets des monnaies et les pays à partir du document XML sur le site de l'ISO.
Pour ne pas relancer l'analyse du document XML des taux de change une deuxième fois dans ces nouvelles méthodes, nous allons déclarer la variable, qui contient la liste des monnaies et leur taux de change, comme attribut d'instance dans la classe du bean. Dans ce cas, il faudrait déclarer le bean session comme stateful : remplacer l'annotation `@Stateless` par `@Stateful`. Ceci permet d'avoir un état (l'objet List) sauvegardé entre les invocations des méthodes du bean. En fait, dans le cas des beans session sans état, le serveur d'application crée un pool de beans, lors du déploiement, et à chaque invocation d'une méthode du bean, il peut prendre

une instance différente, d'où la perte potentielle de l'état enregistré entre deux invocations. Cette solution (où un seul objet répond à plusieurs invocations) est déconseillée dans les applications qui doivent passer à l'échelle. L'idée dans ce sujet de TP est juste de pratiquer un peu l'autre variante de session beans (les *stateful*).

- Modifier index.jsp dans le composant Web pour ne proposer dans l'interface Web que les monnaies dont les taux de change sont disponibles (en s'appuyant sur la méthode getAvailableCurrencies() décrite ci-dessus)
- Créer le composant MDB (MailerMDB) et la file de messages :
  - Cliquer sur l'icône du module EJB ConverterEJB de l'application avec le bouton droit, puis choisir New... > Message Bean ...
  - Donner un nom à votre bean : MailerMDB
  - Donner un nom de package à la classe du MDB (même nom que le bean session)
  - Donner un nom à la classe, puis valider
  - Remplacer l'annotation `@MessageDrive(...)` par :

```
@MessageDriven(activationConfig = {
  @ActivationConfigProperty(
  propertyName = "destination",
  propertyValue = "jms/MailContentQueue"),
  @ActivationConfigProperty(
  propertyName = "destinationType",
  propertyValue = "javax.jms.Queue")
})
```

  - Nous allons maintenant créer la file de message depuis l'interface Web d'administration/management de WildFly (un service de ce type est démarré sur le port 9990. Vérifier le numéro de port dans le fichier de configuration standalone-full.xml : jboss.management.http.port). Aller sur votre navigateur Web sur localhost :9990.
  - Si un message d'erreur s'affiche, vous demandant de créer un compte admin, revenez sur un terminal et aller dans le répertoire WildFly_Home/bin
  - Taper la commande ./add-user.sh, et suivre les étapes pour créer un "*management user*"
  - Donner un login et un mot de passe et laisser les autres paramètres par défaut
  - Revenir sur le navigateur Web et retaper l'URL localhost :9990 puis saisir le login et mot de passe. Vous avez maintenant accès à l'interface Web d'administration du serveur d'application
  - Aller dans l'onglet Configuration, puis choisir Subsystems > Messaging > Server > default > Destinations, et cliquer sur le bouton view
  - Choisir JMS Queue dans le menu à gauche
  - Cliquer sur le bouton Add en haut à droite
  - Donner un nom à la file de messages : `MailContent`
  - Dans « Entries », mettez le nom JNDI : queue/MailContent et surtout ne pas oublier de **valider sur Entrer**
  - Cliquer sur le bouton Add. Le message « JMS Queue MailContent successfully added » doit s'afficher sur la page Web
  - Maintenant la file de messages est créée et enregistrée dans l'annuaire avec le nom JNDI :

queue/MailContent
— Nous allons éditer la classe du bean (MailerMDB.java) plus tard

- Dans le script JSP, ajouter l'envoi de message dans la file créée ci-dessus : après le code permettant la récupération de l'adresse email, ajouter le code suivant (ce code a besoin de quelques imports javax.naming.*, javax.jms.*, ...) :
  — Récupérer le contexte initial dans le serveur de noms JNDI :

```
Context jndiContext = new InitialContext();
```

  — Obtenir une instance de la fabrique de connexions qui a été créée précédemment :

```
javax.jms.ConnectionFactory connectionFactory =
  (QueueConnectionFactory)jndiContext.lookup("/ConnectionFactory");
```

  — Créer une connexion à l'aide de la fabrique de connexions :

```
Connection connection = connectionFactory.createConnection();
```

  — Créer une session sans transactions et avec des accusés de réception :

```
Session sessionQ =
  connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
```

  — Créer un message de type texte utilisant l'objet session :

```
TextMessage message = sessionQ.createTextMessage();
```

  — Mettre le texte nécessaire dans ce message :

```
message.setText(amount+"#"+email);
```

  — Obtenir une référence vers une instance de la file de messages :

```
javax.jms.Queue queue =
  (javax.jms.Queue) jndiContext.lookup("queue/MailContent");
```

  — Créer un objet de type producteur de messages sur la file de messages à l'aide de l'objet session :

```
MessageProducer messageProducer=sessionQ.createProducer(queue);
```

  — Envoyer le message à l'aide de cet objet producteur de messages :

```
messageProducer.send(message);
```

- Éditer la classe du MDB :
  — Ajouter un attribut qui référencera le bean session :

```
@EJB
IConverter converter;
```

  La valeur de cet attribut est injecté automatiquement par le serveur d'application lors du déploiement (donc connexion automatique entre MDB et bean session)

— Selon si votre projet est construit avec Maven ou Gradle, ajouter l'une des deux dépendances suivantes dans le script de build (pom.xml ou build.gradle) :
Pour Maven :

```
<dependency>
  <groupId>javax.mail</groupId>
  <artifactId>mail</artifactId>
  <version>1.4.7</version>
</dependency>
```

Pour Gradle :

```
implementation group: 'javax.mail', name: 'mail', version: '1.4.7'
```

— Importer les classes des packages javax.mail, javax.mail.internet, javax.jms et java.util
— Utiliser un compte Mail à vous (les paramètres SMTP doivent être récupérés auprès de votre fournisseur de service de messagerie. Pour Gmail, voir ci-dessous. Il faudrait également pour Gmail activer, dans les paramètres de configuration, l'envoi de mails à partir d'applications "Less Secure Apps").
— Dans le corps de la méthode onMessage(...) du bean, compléter ce qui suit avec le code nécessaire à la mise en forme des résultats et indiquer votre compte Mail et votre mot de passe :

```
try {
 if (message instanceof TextMessage) {
  TextMessage mesg = (TextMessage) message;
  String content = mesg.getText();
  // Recuperer le montant a convertir :
  String s = content.substring(0,content.indexOf("#"));
  double amount = Double.parseDouble(s);
  // Demander au bean session de faire toutes les conversions ...
  Map<Monnaie,Double> map = converter.euroToOtherCurrencies(amount);
  Properties p = new Properties();
  p.put("mail.smtp.host", "smtp.gmail.com");
  p.put("mail.smtp.auth", "true");
  p.put("mail.smtp.starttls.enable","true");
  javax.mail.Session session = javax.mail.Session.getInstance(p);
  javax.mail.Message msg = new MimeMessage(session);
  try {
    // Preparation du mail
    msg.setFrom(new InternetAddress("<user>@gmail.com"));
    String dest = content.substring(content.indexOf("#")+1);
    msg.setRecipient(javax.mail.Message.RecipientType.TO,
                     new InternetAddress(dest));
    String sujet = "Conversions de monnaie";
    msg.setSubject(sujet);
    // Mettre en forme les resultats retournes par le bean session (Map)
    // dans une chaine de caracteres contenant les balises HTML
    // necessaires pour construire le tableau HTML (variable content)
    // Voir la capture d'ecran de la Figure 1
    msg.setContent(content,"text/html;charset=utf8");
```

Currency	Actual Rate	Converted Amount
SEK(Swedish Krona)	10.2705	1027.0500183105469
BGN(Bulgarian Lev)	1.9558	195.58000564575195
HRK(Kuna)	7.427	742.7000045776367
JPY(Yen)	128.2	12819.999694824219
PLN(Zloty)	4.2968	429.6800136566162
CAD(Canadian Dollar)	1.4873	148.73000383377075
PHP(Philippine Peso)	59.973	5997.299957275391
KRW(Won)	1280.12	128011.99951171875
THB(Baht)	37.298	3729.800033569336
NZD(New Zealand Dollar)	1.6699	166.98999404907227
RON(Romanian Leu)	4.6575	465.74997901916504
BRL(Brazilian Real)	4.2217	422.1700191497803
MYR(Malaysian Ringgit)	4.7133	471.33002281188965
NOK(Norwegian Krone)	9.5378	953.7799835205078

FIGURE 1 – Capture d'écran du courrier électronique envoyé à l'utilisateur

```
    msg.setSentDate(Calendar.getInstance().getTime());
    // Preparation de l'envoi du mail
    Transport transport = session.getTransport("smtp");
    transport.connect("smtp.gmail.com",587,"<utilisateur>","<mot-de-passe>");
    // Envoi du mail
    transport.sendMessage(msg,msg.getAllRecipients());
    transport.close();
    System.out.println("Email envoye a "+dest);
   }
   catch(MessagingException e){e.printStackTrace();}
  }
} catch (JMSException ex) {ex.printStackTrace();}
```

Les noms d'utilisateur et mot de passe doivent être stockées dans des variables d'environnement, de préférence. Ensuite, il faudra les chercher dans le code ci-dessous, en utilisant : `System.getenv(String name)`. Pense à modifier la config de l'exécution (Run Configuration) pour ajouter les deux variables d'environnement. Comme ça, si jamais le code est mis sur un dépôt de code (Git par ex) ou rendu pour une revue par un prof (sur Moodle par ex :-)) ces informations sensibles ne seront pas visibles.

- Déployer et tester votre application (le message du println ci-dessus s'affichera sur la Console de votre IDE)
- Pour re-dépolyer après modification, aller dans l'onglet Services, puis dans votre serveur, cliquer avec le bouton droit de la souris sur l'application Converter et choisir Rerun
- Ne pas oublier de dé-déployer l'application et arrêter le serveur d'application à la fin de la séance