Examen JEE

A l'aide des deux projets Maven fournis, vous complèterez le code en fonction des instructions ci-dessous. La répartition des points est indiquée entre parenthèse à la fin du titre de chaque sous partie. Les deux parties sont indépendantes les unes des autres.

A la fin de l'examen, vous enverrez par mail, un fichier ZIP nommé `exam-jee-NOM-PRENOM.zip` contenant les deux projets complétés. Veillez à supprimer le répertoire `target` de chaque projet afin que le fichier généré ne pèse que quelques dizaine de Ko. Le mail est à adresser à `dmetzler+isen@gmail.com` en ajoutant Cédric Dinont `cedric.dinont@isen-lille.fr` en copie. Il devra contenir votre nom complet dans le body (essayez de personnaliser le corps du mail pour éviter le classement en spam).

Validation de carte de crédit

Dans cette première partie, l'objectif est de créer un objet permettant de valider des numéros de cartes de crédit.

Description de l'algorithme

L'algorithme que l'on utilisera est relativement simple. Les caractéristiques d'une carte de crédit sont les suivantes :

- un numéro formé d'un quadruplet de nombre entre 1 et 9999 séparé par dest tirets (par ex : 1111-2222-3333-4444)
- Les nombres sont à taille fixe et préfixés par des 0 si besoin (ex : 0001)
- un date de validité (mois + année)

On dispose aussi d'un numéro appelé CCV qui permettra de valider la carte. Pour valider un carte :

- on somme les 4 nombres de la carte (1111 + 2222 + 3333 + 4444 = 11110)
- on ajout le mois et l'année (10/2014 -> 11110 +10 + 2014 =13134)
- on multiplie ce nombre par un `SALT` qui vaudra 55 pour tout le reste de l'exercice. (13134 * 55 = 772370)
- on prend le résultat du reste de la division euclidienne (modulo) de ce nombre par le nombre maximum du code CCV. Dans le cas d'une carte VISA, le maximum est 1000, dans le case d'une AMEX c'est 10000. (Visa -> 370, AMEX -> 2370)
- Si le résultat et le CCV donné sont égaux, alors la carte est validée.

Ecriture des tests et implémentation (/7)

En utilisant l'API `org.isen.jee.billing.api`, vous écrirez les différents tests permettant de vérifier le fonctionnement d'un validateur (`org.isen.jee.billing.api.CCValidator`) que vous implémenterez ensuite.

Le test doit vérifier les cas nominaux, mais aussi les cas aux limites (numéros mal formés,

dates incorectes etc...).

Exposition du service via une servlet (/3)

A l'aide de la classe de test `org.isen.jee.billing.ServletTest` vous ferez en sorte d'écrire une servlet permettant de répondre aux requêtes testées.

Attention, les méthodes de test déjà présentes ne doivent pas être modifiées, en revanche vous pourrez ajouter vos propres méthodes de test dans la classe.

Pour activer votre servlet, vous pourrez soit utiliser la déclaration dans le fichier `web.xml`, soit décommenter les lignes 46 et 47 de la classe `org.isen.jee.billing.JettyHarness` et utiliser les annotations.

Interface web de validation (/3)

A l'aide d'une page `JSP`, vous présenterez une interface web permettant de remplir un formulaire contenant les différentes informations d'une carte de crédit.

A la validation de celui-ci, un message s'affiche pour dire si la carte est valide ou non.

Si une carte a déjà été validée, le formulaire sera déjà rempli avec les informations de la carte, sans le CCV. On utilisera pour cela une ou plusieurs variables de session.

Note sur les outils

Pour cette partie, on dispose d'un classe `JettyHarness` qui embarque un conteneur de servlet embarqué. La servlet sera donc testable directement dans le test.

En alternative, on pourra aussi en ligne de commande lancer la commande

```
mvn jetty:run
```

Qui permet de lancer Jetty avec le projet embarqué. On y accèdera par exemple via l'adresse :

```
http://localhost:8080/cc?ccNumber=0001-0000-0000-
0000&month=10&year=2014&ccv=375
```

Compte bancaire

Persistence de données (/10)

Dans cette partie, on s'intéressera à persister des informations de compte bancaire ainsi que les transactions associées dans une base de données. On utilisera pour cela deux objets avec les propriétés suivantes :

- `Account` :

    - ownerName : Nom du client associé au compte
    - balance : Etat du compte
    - ccNumber: numéro de carte de crédit
    - transcations : la liste des transactions
- `Transaction`:
    - type : Débit ou Crédit
    - description: la description de la transaction
    - amount: montant de la transaction
    - date: date de la transaction

A l'aide de l'API fournie et de JPA, faites en sorte que les tests présents dans la classe `org.isen.jee.billing.AccountTest` passent au vert.

Web service (/7)

En implémentant un service de type JAX-RS, faites en sorte que les tests de la classe `org.isen.jee.billing.AccountServiceTest` passent au vert.

Note sur les outils

Pour cette partie, on dispose d'un classe `ContainerHarness` qui embarque un conteneur d'EJB embarqué. Les tests embarquent donc tout le nécessaire pour démarrer un mini serveur d'applicaiton embarqué.

On pourra aussi en ligne de commande lancer les commandes suivantes :

```
mvn clean install
mvn tomee:run
```

Qui permettent de lancer TomEE avec le projet embarqué. L'API sera alors disponible à l'adresse suivante :

```
http://localhost:8080/2-jpa/api/
```