Projet 9

«Testez vos développements Java»

Présentation

Version 1.0

Dubois Geoffrey

analyste-programmeur

Projet 9 : *«Testez vos développements Java»*

1- Les erreurs

2- Implémentation des fonctionnalités

- 2.1- TODO
  - 2.1.1- AddReference()
  - 2.1.2- checkEcritureComptableUnit()
  - 2.1.3- Respect des régles de gestion
- 2.2- Implémentations requises au projet

3- Les Tests

- 3.1- Test unitaire
  - 3.1.1- Couche Model
  - 3.1.2- Couche Business
- 3.2- Test d'intégration
  - 3.2.1- Couche Business
  - 3.2.2- Couche Consumer

Couche Model, Classe EcritureComptable, Correction d'erreur sur la méthode **getTotalCredit()** :
1- la méthode récupérait le **débit** de l'écriture comptable au lieu du **crédit. getDebit()** ==> **getCredit()**

Couche Model, Classe EcritureComptable, Correction d'erreur sur la méthode **isEquilibree()** :
1- Une erreur dans la comparaison était présente. **equals()** ==> **compareTo()**

Couche Model, Classe EcritureComptable, Correction d'erreur sur l'attribut **reference** :
1- Le format de l'attribut reference était incorrect. Le **code journal** prenait en compte des **chiffres** au lieu de l'**ensemble des caractères.** (regexp = «**d**{1,5}-\\d{4}/\\d{5}») ==> (regexp = «**w**{1,5}-\\d{4}/\\d{5}»)

Couche Model, Classe SequenceEcritureComptable, Correction d'erreur sur l'attribut **journalCode** :
1- L'attribut était absent. Ajout de l'attribut «**private String journalCode;**», de son **getter** , de son **setter** et de son remplissage dans le **constructeur**

Couche Business, Classe ComptabiliteManagerImpl, Correction d'erreur sur la méthodes **updateEcritureComptable()** :
1- la **vérification** de l'écriture comptable n'était pas effectué **avant la mise à jour** de l'écriture comptable. this.**checkEcritureComptable(pEcritureComptable);** avant getDaoProxy().getComptabiliteDao().**updateEcriture-Comptable(pEcritureComptable);**

Couche Consumer, fichier sqlContext.xml, Correction d'erreur sur la requête **SQLinsertListLigneEcritureComptable** :
1- une virgule était absente dans la requête d'insertion.

```
<value>
      INSERT INTO myerp.ligne_ecriture_comptable (
                        ecriture_id, ligne_id, compte_
comptable_numero, libelle, debit credit
                  )
      VALUES (
                     :ecriture_id, :ligne_id, :compte_
comptable_numero, :libelle, :debit, :credit
                  )
</value>
```

==>

```
<value>
      INSERT INTO myerp.ligne_ecriture_comptable (
                        ecriture_id, ligne_id, compte_
comptable_numero, libelle, debit, credit
                  )
      VALUES (
                     :ecriture_id, :ligne_id, :compte_
comptable_numero, :libelle, :debit, :credit
                  )
</value>
```

2.1- TODO

2.1.1- AddReference()

```java
@Override
public synchronized void addReference(EcritureComptable pEcritureComptable) {
    int annee = Year.now().getValue();
    String journalCode = pEcritureComptable.getJournal().getCode();
    StringBuilder reference = new StringBuilder();

    try{
        SequenceEcritureComptable sequenceEcritureComptable = getDaoProxy().getComptabiliteDao().getSequenceEcritureComptable(journalCode,annee);
        sequenceEcritureComptable.setDerniereValeur(sequenceEcritureComptable.getDerniereValeur() + 1);
        reference.append(journalCode)
                .append("-")
                .append(annee)
                .append("/")
                .append(StringUtils.leftPad(String.valueOf(sequenceEcritureComptable.getDerniereValeur()), size: 5, padStr: "0"));
        pEcritureComptable.setReference(reference.toString());
        updateSequenceEcritureComptable(sequenceEcritureComptable);
    }catch (NotFoundException e){
        SequenceEcritureComptable sequenceEcritureComptable = new SequenceEcritureComptable(journalCode,annee, pDerniereValeur: 1);
        reference.append(journalCode)
                .append("-")
                .append(annee)
                .append("/")
                .append(StringUtils.leftPad(String.valueOf(1), size: 5, padStr: "0"));
        pEcritureComptable.setReference(reference.toString());
        insertSequenceEcritureComptable(sequenceEcritureComptable);
    }
}
```

2.1.2- checkEcritureComptableUnit()

RG_Compta_5	La référence d'une écriture comptable est composée du code du journal dans lequel figure l'écriture suivi de l'année et d'un numéro de séquence (propre à chaque journal) sur 5 chiffres incrémenté automatiquement à chaque écriture. Le formatage de la référence est : XX-AAAA/#####. Ex : Journal de banque (BQ), écriture au 31/12/2016 --> BQ-2016/00001

```
    String reference = pEcritureComptable.getReference();
    final String regexp = "\\w{2}-\\d{4}/\\d{5}";
    if(!reference.matches(regexp))
        throw new FunctionalException("La référence n'a pas le bon pattern XX-AAAA/#####");
    int annee = Integer.parseInt(StringUtils.substringBetween(reference, open: "-", close: "/"));
    if(pEcritureComptable.getDate().toInstant().atZone(ZoneId.systemDefault()).getYear() != annee)
        throw new FunctionalException("L'annee de la référence n'est pas égale");
    String code = StringUtils.substringBefore(reference, separator: "-");
    if(!pEcritureComptable.getJournal().getCode().equals(code))
        throw new FunctionalException("Le code de la référence n'est pas égale");
}
```

2.1.3- Respect des régles de gestion

```
@Override
public void checkEcritureComptable(EcritureComptable pEcritureComptable) throws FunctionalException {
    this.checkEcritureComptableUnit(pEcritureComptable);
    this.checkEcritureComptableContext(pEcritureComptable);
}
```

RG_Compta_2 Pour qu'une écriture comptable soit valide, elle doit être équilibrée : la somme des montants au crédit des lignes d'écriture doit être égale à la somme des montants au débit.

```
        if (!pEcritureComptable.isEquilibree()) {
            throw new FunctionalException("L'écriture comptable n'est pas équilibrée.");
        }
```

RG_Compta_3 Une écriture comptable doit contenir au moins deux lignes d'écriture : une au débit et une au crédit.

```
int vNbrCredit = 0;
int vNbrDebit = 0;
for (LigneEcritureComptable vLigneEcritureComptable : pEcritureComptable.getListLigneEcriture()) {
    if (BigDecimal.ZERO.compareTo(ObjectUtils.defaultIfNull(vLigneEcritureComptable.getCredit(),
                                                            BigDecimal.ZERO)) != 0) {
        vNbrCredit++;
    }
    if (BigDecimal.ZERO.compareTo(ObjectUtils.defaultIfNull(vLigneEcritureComptable.getDebit(),
                                                            BigDecimal.ZERO)) != 0) {
        vNbrDebit++;
    }
}
if (pEcritureComptable.getListLigneEcriture().size() < 2
    || vNbrCredit < 1
    || vNbrDebit < 1) {
        throw new FunctionalException(
            "L'écriture comptable doit avoir au moins deux lignes : une ligne au débit et une ligne au crédit.");
    }
```

RG_Compta_6 La référence d'une écriture comptable doit être unique, il n'est pas possible de créer plusieurs écritures ayant la même référence.

```
    if (StringUtils.isNoneEmpty(pEcritureComptable.getReference())) {
        try {
            EcritureComptable vECRef = getDaoProxy().getComptabiliteDao().getEcritureComptableByRef(
                pEcritureComptable.getReference());

            if (pEcritureComptable.getId() == null
                || !pEcritureComptable.getId().equals(vECRef.getId())) {
                throw new FunctionalException("Une autre écriture comptable existe déjà avec la même référence.");
            }
        } catch (NotFoundException vEx) {
        }
    }
}
```

2.2- Implémentations requises au projet

Couche Consumer, Classe ComptabiliteDao, **getSequenceEcritureComptable()** :
insertSequenceEcritureComptable() :
updateSequenceEcritureComptable() :
deleteSequenceEcritureComptable() :

1- **Création** des **signatures** des méthodes dans l'**interface ComptabiliteDao.**

Couche Consumer, Classe ComptabiliteDaoImpl, **getSequenceEcritureComptable()** :
insertSequenceEcritureComptable() :
updateSequenceEcritureComptable() :
deleteSequenceEcritureComptable() :

1- **Création** des méthodes dans **ComptabiliteDaoImpl** en **@Override.**
2- **Implémentation** des méthodes.
3- **Création** des méthode **setSQL** associée et de l'attribut statique **SQL** associé.

Couche Business, package testbusiness.business, Ajout de fichier **bootstrapContext.xml** :

1- **Erreur survenu** : impossible d'ouvrir le fichier bootStrapContext.xml
2- **Configuration** : Importation des fichiers de configurations.

Couche Consumer, Classe SequenceEcritureComptableRM, Création de la classe **SequenceEcritureComptableRM** :

1- **mapRow()** : Permet de mapper les données du resultSet au bean

3.1- Test unitaire

3.1.1- Couche Model

```
    Test les getters et les setters
@Test
public void validateSettersAndGetters() {
    final PojoClass EcritureComptablePojo = PojoClassFactory.getPojoClass(EcritureComptable.class);

    final Validator validator = ValidatorBuilder.create()
            .with(new SetterTester(), new GetterTester())
            .build();
    validator.validate(EcritureComptablePojo);
}

    Test de la fonction isEquilibre d'une écriture comptable renvoie true
@Test
public void isEquilibree() {
    EcritureComptable vEcriture;
    vEcriture = new EcritureComptable();

    vEcriture.setLibelle("Equilibrée");
    vEcriture.getListLigneEcriture().add(this.createLigne( pCompteComptableNumero: 1,  pDebit: "200.50",  pCredit: null));
    vEcriture.getListLigneEcriture().add(this.createLigne( pCompteComptableNumero: 1,  pDebit: "100.50",  pCredit: "33"));
    vEcriture.getListLigneEcriture().add(this.createLigne( pCompteComptableNumero: 2,  pDebit: null,  pCredit: "301"));
    vEcriture.getListLigneEcriture().add(this.createLigne( pCompteComptableNumero: 2,  pDebit: "40",  pCredit: "7"));
    Assert.assertTrue(vEcriture.toString(), vEcriture.isEquilibree());
}
```

```
Test de la fonction getTotalCredit vérifie que le chiffre renvoie le résultat attendu
@Test
public void getTotalCredit_returnBigDecimal_EcritureComptable(){
    EcritureComptable vEcriture;
    vEcriture = new EcritureComptable();
    vEcriture.getListLigneEcriture().clear();
    vEcriture.setLibelle("check total debit");
    vEcriture.getListLigneEcriture().add(this.createLigne( pCompteComptableNumero: 1,  pDebit: "10",  pCredit: null));
    vEcriture.getListLigneEcriture().add(this.createLigne( pCompteComptableNumero: 1,  pDebit: "20",  pCredit: "10.20"));
    vEcriture.getListLigneEcriture().add(this.createLigne( pCompteComptableNumero: 1,  pDebit: "300",  pCredit: null));
    vEcriture.getListLigneEcriture().add(this.createLigne( pCompteComptableNumero: 1,  pDebit: "150",  pCredit: "21"));
    vEcriture.getListLigneEcriture().add(this.createLigne( pCompteComptableNumero: 2,  pDebit: null,  pCredit: "302.9"));
    vEcriture.getListLigneEcriture().add(this.createLigne( pCompteComptableNumero: 2,  pDebit: "1.40",  pCredit: "2"));
    vEcriture.getListLigneEcriture().add(this.createLigne( pCompteComptableNumero: 2,  pDebit: null,  pCredit: "179"));
    vEcriture.getListLigneEcriture().add(this.createLigne( pCompteComptableNumero: 2,  pDebit: "1.0",  pCredit: "271"));
    BigDecimal result = BigDecimal.valueOf(786.1);

    Assert.assertTrue( condition: vEcriture.getTotalCredit().compareTo(result) == 0);
}
```

>>> Exécution des tests unitaires de la couche Model

3.1.1- Couche Business

```
Test de addReference
Throws: NotFoundException – FunctionalException
@Test
public void addReference_NoReturn_EcritureComptable() throws NotFoundException {
    EcritureComptable ecritureComptable = new EcritureComptable();
    ecritureComptable.setDate(new Date());
    ecritureComptable.setId(21);
    ecritureComptable.setLibelle("Ecriture comptable test add reference");
    ecritureComptable.setJournal(new JournalComptable( pCode: "VE", pLibelle: "test"));
    ecritureComptable.getListLigneEcriture().add(
            new LigneEcritureComptable(
                    new CompteComptable( pNumero: 1), pLibelle: null,new BigDecimal( val: 42)
                    , pCredit: null
            ));
    ecritureComptable.getListLigneEcriture().add(
            new LigneEcritureComptable(
                    new CompteComptable( pNumero: 2), pLibelle: null, pDebit: null
                    ,new BigDecimal( val: 42)
            ));

    SequenceEcritureComptable sequenceEcritureComptable = new SequenceEcritureComptable( pJournalCode: "VE", pAnnee: 2020, pDerniereValeur: 16);

    DaoProxy daoProxy = Mockito.mock(DaoProxy.class);
    ComptabiliteDao comptabiliteDao = Mockito.mock(ComptabiliteDao.class);
    TransactionManager transactionManager = Mockito.mock(TransactionManager.class);

    Mockito.when(daoProxy.getComptabiliteDao()).thenReturn(comptabiliteDao);
    Mockito.when(comptabiliteDao.getSequenceEcritureComptable( pJournalCode: "VE", pAnnee: 2020)).thenReturn(sequenceEcritureComptable);

    AbstractBusinessManager.configure( pBusinessProxy: null,daoProxy,transactionManager);
    comptabiliteManager.addReference(ecritureComptable);

    Assert.assertEquals(ecritureComptable.toString(), expected: "VE-2020/00017",ecritureComptable.getReference());
}
```

```java
/** Test de la fonction checkEcritureComptableUnit ...*/
@Test(expected = FunctionalException.class)
public void checkEcritureComptableUnitRG2_NoEquilibre_ThrowFunctionalException() throws Exception {...}

/** Test de la fonction checkEcritureComptableUnit ...*/
@Test
public void checkEcritureComptableUnitRG2_Equilibre_DontThrowException() throws Exception {...}

/** Test de la fonction checkEcritureComptableUnit ...*/
@Test(expected = FunctionalException.class)
public void checkEcritureComptableUnitRG3_WithTwoDebit_throwFunctionalException() throws Exception {...}

/** Test de la fonction checkEcritureComptableUnit ...*/
@Test(expected = FunctionalException.class)
public void checkEcritureComptableUnitRG3_WithOneLine_throwFunctionalException() throws Exception {...}

/** Test de la fonction checkEcritureComptableUnit ...*/
@Test(expected = FunctionalException.class)
public void checkEcritureComptableUnitRG3_WithTwoCredit_throwFunctionalException() throws Exception {...}

/** Test de la fonction checkEcritureComptableUnit ...*/
@Test(expected = FunctionalException.class)
public void checkEcritureComptableUnitRG3_WithThreeCredit_throwFunctionalException() throws Exception {...}

/** Test de la fonction checkEcritureComptableUnit ...*/
@Test
public void checkEcritureComptableUnitRG3AndRG2_WithThreeLine_NoThrowFunctionalException() throws Exception {...}


/** Test de la fonction checkEcritureComptableUnit ...*/
@ParameterizedTest(name = "{0} bad référence throw FunctionalException")
@ValueSource(strings = {"AC-20m0/00001","ACA-2020/00001","AC-200/000X1","AC-2020/000018","AC-2020/0018"})
public void checkEcritureComptableUnitRG5_BadFormat_ThrowFunctionalException(String ref){...}
```

>>> Exécution des tests unitaires de la couche Business

3.2- Test d'intégration

3.2.1- Couche Business

```
Test de mise à jour d'une écriture comptable. Insertion d'une écriture, puis modification. Vérification de la
modification dans la bdd suppression de l'écriture insérée
@Test
public void updateEcritureComptableTest_NoReturn_checkEcritureComptable() throws Exception {

    SimpleDateFormat simpleDateFormat = new SimpleDateFormat( pattern: "yyyy/MM/dd hh:mm:ss");
    Date date = simpleDateFormat.parse( source: "2020/09/18 00:00:00");

    JournalComptable journalComptable = new JournalComptable();
    JournalComptable journalComptable1 = new JournalComptable();

    List<JournalComptable> listJournalComptable = comptabiliteManager.getListJournalComptable();
    for (JournalComptable journal : listJournalComptable) {
        if (StringUtils.equals(journal.getCode(), "AC")) {
            journalComptable = journal;
        }
        if (StringUtils.equals(journal.getCode(), "BQ")) {
            journalComptable1 = journal;
        }
    }

    CompteComptable compteComptable = comptabiliteManager.getListCompteComptable().get(0);

    LigneEcritureComptable ligneEcritureComptable = new LigneEcritureComptable(compteComptable, pLibelle: "Ligne 1", BigDecimal.valueOf(200), pCredit: null);
    LigneEcritureComptable ligneEcritureComptable1 = new LigneEcritureComptable(compteComptable, pLibelle: "Ligne 2", pDebit: null, BigDecimal.valueOf(200));
    List<LigneEcritureComptable> listLigneEcriture = new ArrayList<>();
    listLigneEcriture.add(ligneEcritureComptable);
    listLigneEcriture.add(ligneEcritureComptable1);

    EcritureComptable ecritureComptable = createEcritureComptable( id: null, journalComptable, reference: "AC-2020/09999", date, libelle: "ecriture insérée", listLigneEcriture);

    comptabiliteManager.insertEcritureComptable(ecritureComptable);

    Date date1 = simpleDateFormat.parse( source: "2020/09/18 00:00:00");

    LigneEcritureComptable ligneEcritureComptable2 = new LigneEcritureComptable(compteComptable, pLibelle: "Ligne 3", BigDecimal.valueOf(500), pCredit: null);
    LigneEcritureComptable ligneEcritureComptable3 = new LigneEcritureComptable(compteComptable, pLibelle: "Ligne 4", pDebit: null, BigDecimal.valueOf(500));
    List<LigneEcritureComptable> listLigneEcriture2 = new ArrayList<>();
    listLigneEcriture2.add(ligneEcritureComptable2);
    listLigneEcriture2.add(ligneEcritureComptable3);

    EcritureComptable ecritureComptableModifiee = createEcritureComptable(ecritureComptable.getId(), journalComptable1, reference: "BQ-2020/10000", date1, libelle: "ecriture modifié", listLigneEcriture2);
    comptabiliteManager.updateEcritureComptable(ecritureComptableModifiee);
```

```
    List<EcritureComptable> listEcritureComptable = comptabiliteManager.getListEcritureComptable();
    Iterator iterator = listEcritureComptable.listIterator();
    boolean testCheck = false;


    while (iterator.hasNext()) {
        EcritureComptable ecriture = (EcritureComptable) iterator.next();
        if (ecriture.getId().equals(ecritureComptableModifiee.getId()) &&
        ecriture.getDate().compareTo(ecritureComptableModifiee.getDate()) == 0 &&
        StringUtils.equals(ecriture.getJournal().getCode(), ecritureComptableModifiee.getJournal().getCode()) &&
        StringUtils.equals(ecriture.getLibelle(), ecritureComptableModifiee.getLibelle()) &&
        StringUtils.equals(ecriture.getReference(), ecritureComptableModifiee.getReference()) &&
        ecriture.getListLigneEcriture().size() == ecritureComptableModifiee.getListLigneEcriture().size() &&
        StringUtils.equals(ecriture.getListLigneEcriture().get(0).getLibelle(), ecritureComptableModifiee.getListLigneEcriture().get(0).getLibelle()) &&
        StringUtils.equals(ecriture.getListLigneEcriture().get(1).getLibelle(), ecritureComptableModifiee.getListLigneEcriture().get(1).getLibelle())) {
            testCheck = true;
        }
    }

    assertTrue( message: "Erreur dans la mise à jour de l'écriture comptable", testCheck);

    comptabiliteManager.deleteEcritureComptable(ecritureComptable.getId());
}
```

>>> Exécution des tests d'intégration de la couche Business

3.2.1- Couche Consumer

```java
/**
 * Test de récupération de la liste des écritures comptables
 * Vérification que la taille de liste est égale à 5
 * et que la dernière écriture comptable vaut :
 * - id = -5
 * - code journal = BQ
 * - reference = BQ-2016/00005
 * - data = 2016/12/27 00:00:00
 * - libellé = Paiement Facture C110002
 * - 2 lignes d'écriture :
 *      - Code compte comptable ligne écriture 2 = 411
 *      - Crédit ligne écriture 2 = 3000
 *
 */
@Test

public void getListEcritureComptableTest_listEcritureComtable_checkFirstEcritureComptableAndSizeListEqual5() throws ParseException {

    List<EcritureComptable> listEcritureComptable = comptabiliteDao.getListEcritureComptable();

    assertTrue( message: "La liste des écritures n'est pas égale à 5",  condition: listEcritureComptable.size() == 5);

    SimpleDateFormat simpleDateFormat = new SimpleDateFormat( pattern: "yyyy/MM/dd hh:mm:ss");
    Date date = simpleDateFormat.parse( source: "2016/12/27 00:00:00");
    boolean testCheck = false;
    if(listEcritureComptable.get(4).getId() == -5 &&
            StringUtils.equals(listEcritureComptable.get(4).getJournal().getCode(), "BQ") &&
            StringUtils.equals(listEcritureComptable.get(4).getReference(), "BQ-2016/00005") &&
            StringUtils.equals(listEcritureComptable.get(4).getLibelle(), "Paiement Facture C110002") &&
            listEcritureComptable.get(4).getDate().compareTo(date) == 0 &&
            listEcritureComptable.get(4).getListLigneEcriture().size() == 2 &&
            listEcritureComptable.get(4).getListLigneEcriture().get(1).getCompteComptable().getNumero() == 411 &&
            listEcritureComptable.get(4).getListLigneEcriture().get(1).getCredit().compareTo(BigDecimal.valueOf(3000)) == 0)
    {
        testCheck = true;
    }
    assertTrue( message: "L'écriture comptable d'ID -5, de code journal BQ, de référence BQ-2016/00005, de date 2016/12/27 00:00:00 et de libellé Paiement Facture C110002 r
}
```

>>> Exécution des tests d'intégration de la couche Consumer

4.1- Le fichier de configuration *.travis.yml*

```yaml
# Fichier de configuration Travis--

language: java
jdk:
  - openjdk8
sudo: false
services:
  - docker
before_script:
  - docker-compose -f docker/dev/docker-compose.yml up -d
script:
  - mvn --file ./src/pom.xml clean jacoco:prepare-agent package jacoco:report coveralls:report -P test-consumer,test-business -e
#  - mvn --file ./src/pom.xml clean test -P test-consumer,test-business
after_script:
  - docker-compose -f docker/dev/docker-compose.yml stop
  - docker-compose -f docker/dev/docker-compose.yml rm -f
```

4.2- Fonctionnement de Travis

4.2.1- Modification d'un fichier du projet

4.2.2- Commit et push avec gitHub

4.2.3- Lancement automatique de travis

4.2.4- Rapport de test

4.2.5- Couverture de code

- Travis exécute, à chaque push, un rapport de test

- Un «code coverage» supérieur à 75%

- Informations disponibles depuis le Readme

Merci de votre attention !

Des questions ?