Rapport de Génie Logiciel

Titouan Loiseau et Baptiste Marchand
4A SAGI - TD2

Sommaire

Introduction

Ceci est notre rapport du cours de génie logiciel lors duquel nous avons découvert et manipulé des technologies telles que git, la modélisation UML ou JUnit. Il suit les questions du dernier TD effectué en duo.

Il est possible de retrouver le projet sur le dépôt GitHub se trouvant à l'addresse suivante: https://github.com/titouan-loiseau/Genie-Logiciel

1 Exercice 1

1.1 Q1

La figure 1 présente le premier diagramme de classes imaginé.

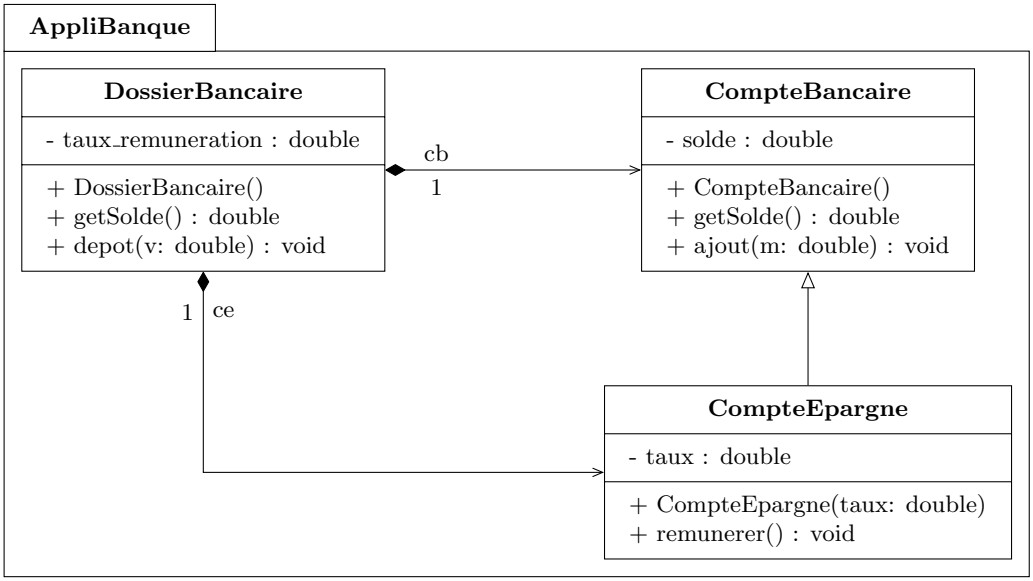

Figure 1: Diagramme de classes

1.2 Q2

La figure 2 présente le premier diagramme de séquences imaginé.

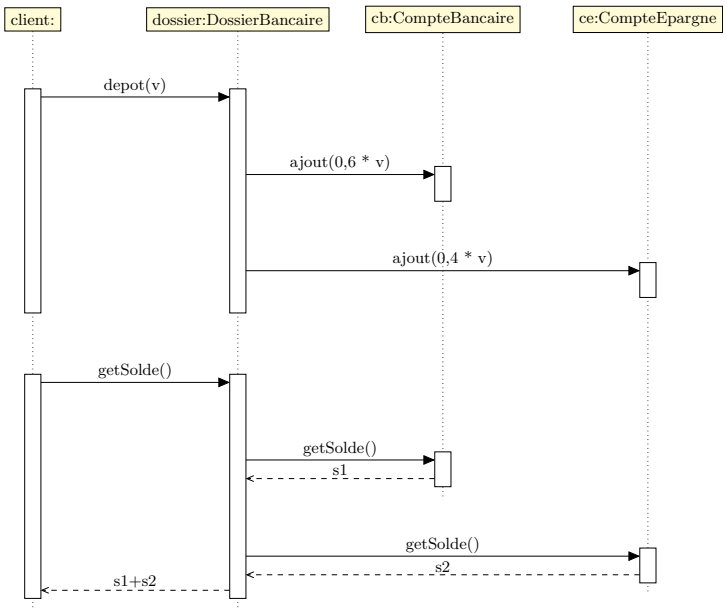

Figure 2: Diagramme de séquence

1.3 Q3

En plus des diagrammes précédents, on peut faire un diragramme d'objets modélisant l'état du programme après l'instanciation de l'objet de la classe DossierBancaire. La figure 3 présente un tel diagramme.

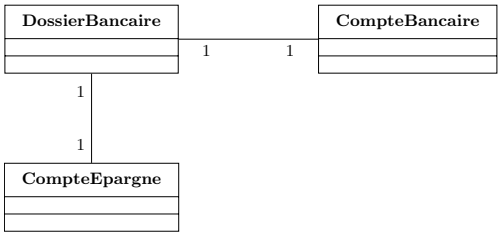

Figure 3: Diagramme d'objets

2 Exercice 2

Pour la compilation du projet simplement, les commandes trouvées dans le README permettent de compiler le projet, comme montré dans la figure 4.

```
javac AppliBanque/Main.java
java AppliBanque/Main
```

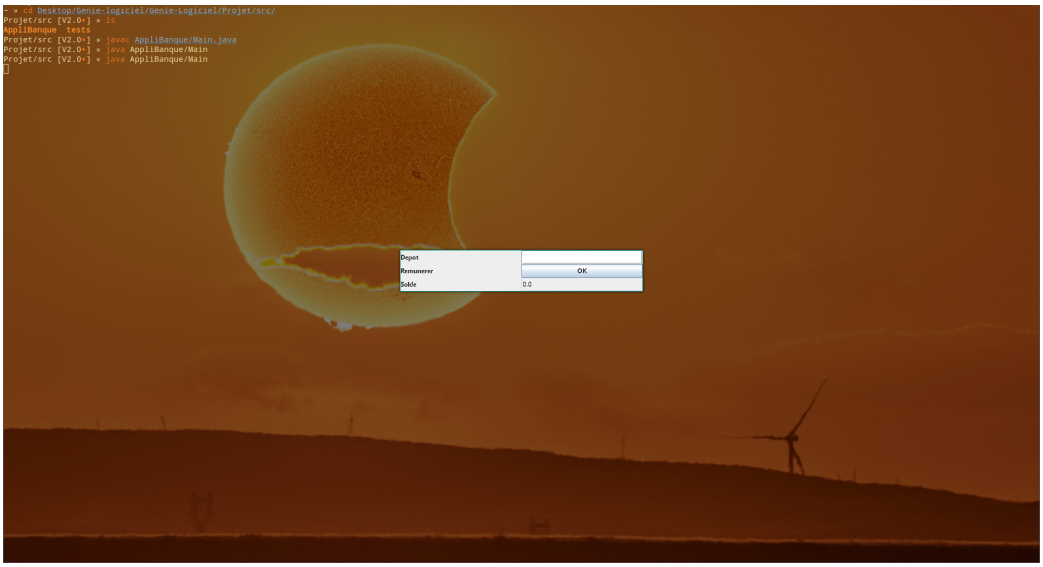

Figure 4: Compilation du programme

Pour la compilation des tests, il faut d'abord télécharger JUnit 4 et hamcrest-core et placer les fichiers .jar à un endroit connu (l'emplacement du projet par exemple). En utilisant les commandes du README en adaptant les chemins en fonction de l'emplacement local, on peut compiler et exécuter les tests.

```
javac --classpath
    "chemin/vers/juint.jar":"chemin/vers/hamcrest.jar":
    tests/TestAppliBanque.java
java --classpath
    "chemin/vers/juint.jar":"chemin/vers/hamcrest.jar":
    tests/TestAppliBanque
```

3 Exercice 3

3.1 Q1-Q2

Pour travailler parallèlement sur le projet, nous avous utilisé le logiciel de versionning git couplé à un outil de dépôt en ligne appelé GitHub. Les étapes effectuées pour initialiser le dépôt sont:

- Création de l'arboresence des fichiers, avec un dossier "Projet" contenant le code Java, et un dossier "Rapport" contenant le rapport
- Ecriture du fichier .gitignore, permettant de ne pas avoir de fichiers inutiles dans le dépôt, notamment le bytecote généré par Java et les fichiers de log de LaTeX.
- Initialisation du dépôt local avec la commande 'git init'
- Ajout des fichiers à prendre en compte dans le commit avec 'git add .'
- Commit la version de départ avec 'git commit -m "Commit initial"'
- Création du dépôt distant sur github avec l'utilitaire invoqué par la commande'gh repo create'
- Push du contenu local sur le dépôt distant avec 'git push -u origin master'

Par la suite, nous avons effectué les différents commits via la console en ajoutant des tags quand nécéssaire avec la commande 'git tag'. Pour ajouter les tags dans le dépôt distant, il faut exécuter la commande 'git push origin –tags'

3.2 Q3

```
// TestDossierBancaire.java
@Test
public void test_constructeur()
{
   DossierBancaire dossier=new DossierBancaire();
   assertNotNull(dossier);
}
```

```java
@Test
public void test_deposer()
{
   DossierBancaire dossier=new DossierBancaire();
   dossier.deposer(100);
   assertEquals(100,dossier.get_solde(),0);
}

@Test
public void test_remunerer()
{
   DossierBancaire dossier=new DossierBancaire();
   dossier.deposer(100);
   dossier.remunerer();
   double apres_remuneration = 0.4 * 100 + 0.6 * 100 * 1.032;
   assertEquals(apres_remuneration,dossier.get_solde(),0);
}
```

L'exécution de ces tests fonctionne sauf pour rémunérer qui n'a pas encore été implémentée, comme montré sur la figure 5.

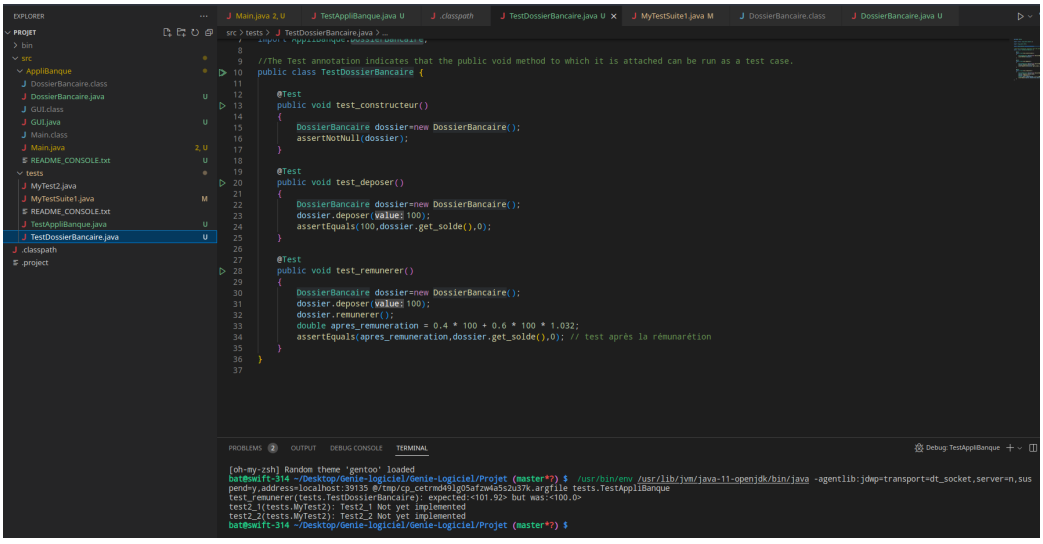

Figure 5: Tests

3.3 Q5

```
// CompteBancaire.java
...
public class CompteBancaire {

    protected double m_solde;

    public CompteBancaire()
    {
        m_solde=0;
    }
    public void deposer(double value) {m_solde+=value;}
    public double get_solde() {return m_solde;}

}
```

```
// DossierBancaire.java
...
public class DossierBancaire {

    private CompteBancaire m_compte_bancaire;

    //Constructeur
    public DossierBancaire()
    {
        m_compte_bancaire = new CompteBancaire();
    }

    public void deposer(double value)
        {m_compte_bancaire.deposer(value);}
    public void getSolde() {return m_compte_bancaire.getSolde();}
    public void remunerer() {}

}
```

3.4 Q6

```
// CompteEpargne.java
...
public class CompteEpargne extends CompteBancaire {

    private double taux;
    public CompteEpargne(double t)
    {
        taux = 1+t/100;
      m_solde=0;
    }

    public void remunerer(){m_solde = m_solde * taux;}
}
```

Lors de l'implémentation de CompteEpargne dans DossierBancaire, nous n'avons pas utilisé de variable taux_remuneration (comme indiqué dans le diagramme UML). A la place, le taux est écrit en dur dans le constructeur de CompteEpargne. La version finale du diagramme de classes est disponible à la figure 6.

```
// DossierBancaire.java
...
public class DossierBancaire {

    private CompteEpargne m_compte_epargne;
    private CompteBancaire m_compte_bancaire;

    //Constructeur
    public DossierBancaire()
    {
        m_compte_bancaire = new CompteBancaire();
        m_compte_epargne = new CompteEpargne(3.2);
    }

    public void deposer(double value) {
        m_compte_bancaire.deposer(value*0.4);
        m_compte_epargne.deposer(value*0.6);
    }
    public void getSolde() {return m_compte_bancaire.getSolde() +
```

```
        m_compte_epargne.getSolde();}
    public void remunerer() {m_compte_epargne.remunerer();}

}
```

Les tests sont déjà adaptés à l'implémentation du compte épargne, et réussisent maintenant car le compte épargne est implémenté.

4 Exercice 4

4.1 Q1

On a rajouté quelques commentaires dans la classe DossierBancaire.

4.2 Q2

Le retour à une version précédente se fait via la commande

```
git checkout V2.0
```

(pour retourner au commit avec le tag V2.0 par exemple)

La création de la nouvelle branche à partir de V2.0 se fait avec la commande

```
git branch new_dev
```

Pour se rendre sur la branche (déplacer la HEAD sur la branche), il faut faire la commande

```
git checkout new_dev
```

4.3 Q3

Pour retourner sur la branche principale (déplacer la HEAD sur la branche), il faut faire la commande

```
git checkout master
```

4.4 Q4

L'intégration des modification apportees dans new_dev sur la branche master se fait au moyen d'un merge. Ce dernier s'effectue avec la commande:

```
git merge new_dev master
```

4.5 Q5

Nous n'avons pas eu d'erreur de fusion, et après exécution le programme fonctionne normalement.

5 Exercice 5

5.1 Q1

L'ajout de la possibilité de retirer de l'argent se fait dans le Compte Bancaire. La vérification du solde se fait de dans Dossier Bancaire.

```
// CompteBancaire.java
...
public void retirer(double value) {m_solde = m_solde - value;}
...
```

```
// DossierBancaire.java
...
public void retirer(double value) throws Exception
{
    if(value > m_compte_bancaire.getSolde()) {
        throw new Exception("Pas assez d'argent sur le compte
            bancaire");
    }else{
        m_compte_bancaire.retirer(value);
    }
}
...
```

5.2 Q2

Les tests unitaires testent le cas où suffisament d'argent et présent et le cas inverse.

```
// TestSuite.java
...
public class TestSuite {

   @Test
   public void test_retirer_suffisant() throws Exception
   {
      DossierBancaire dossier=new DossierBancaire();
      dossier.deposer(100);
      dossier.retirer(20);
      assertEquals(80,dossier.get_solde(),0);
   }

   @Test
   public void test_retirer_insuffisant()
   {
      DossierBancaire dossier=new DossierBancaire();
      dossier.deposer(100);
      try {
         dossier.retirer(41);
      } catch (Exception e) {
         //e.printStackTrace();
         fail("Insuffisance dans le compte Bancaire");
      }

   }

}
```

5.3 Q3

Pour le GUI, on a recopié et réutilisé le code existant pour rajouter un bouton et un text field qui permettent d'appeler la fonction retirer.

6 Conclusion

Lors du projet, nous avons implémenté et programmé d'une façon qui diffère du modèle proposé dans le diagramme de classe décrit à la figure 1. Finalement, notre programme peut être modélisé selon le diagramme de classes de la figure 6.

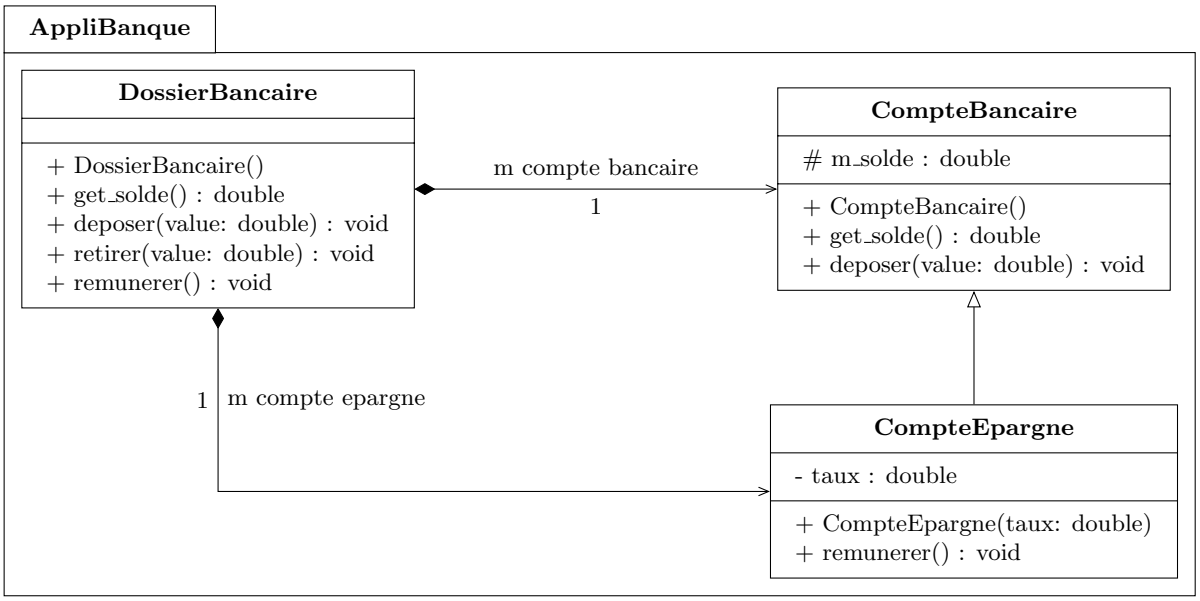

Figure 6: Diagramme de classes final

Le rapport du TD a été réalisé sur une branche différente du projet qui a été fusionnée à la toute fin.