



GÉNIE LOGICIEL : MAXIME DOYEN - ROMAIN
FRESNEAU

Projet Compte Bancaire

Equipe :
ROMAIN FRESNEAU
MAXIME DOYEN

Encadrants :
M.DELANOUE

2022-2023

Contents

1	Analyse fonctionnelle	2
1.1	Diagramme UML du projet	2
1.2	Diagramme UML de séquences	4
1.3	Diagramme d'objets et diagramme de cas d'utilisation	4
2	Code de départ	5
2.1	Exercice2	5
3	Développement: Exercice3	7
3.1	Question 1 et 2	7
3.2	Question 3	8
3.3	Question 4	9
3.4	Question 5	9
3.5	Question 6	10
3.6	Question 7	11
3.7	Question 8	11
3.8	Question 9	12
4	Fusion : Exercice4	12
4.1	Question 1	12
4.2	Question 2	12
4.3	Question 3	12
4.4	Question 4	13
4.5	Question 5	13
5	Tests : Exercice5	13
5.1	Question 1	13
5.2	Question 2	14
5.3	Question 3	15
6	Conclusion	15

1 Analyse fonctionnelle

Premi re r flexion sur la structure de notre projet : Il faut r aliser l'analyse fonctionnelle du projet. Pour commencer, il faut d cider de la structure du projet. Apr s avoir lu le cahier des charges, il semble  vident qu'il faut que notre projet contienne :

- D finir la classe CompteBancaire
- D finir la classe CompteEpargne
- D finir la classe ComptePayant

Ensuite, nous avons r fl chi et il semble plus pratique de cr er une quatri me classe Client qui est compos  d'un compte bancaire. C'est une composition forte puisque si le compte est supprim  alors le client n'est plus consid r  comme client.

Par la suite il faut d finir une classe contenant la fonction main() permettant de tester les classes.

Pour la structure de notre diagramme UML, nous avons d cider de cr er les trois classes pr sentes dans le texte d' nonc . La Classe dossier bancaire est compos  du compte  pargne et du compte courant. Si les comptes disparaissent alors le dossier bancaire est supprim  aussi.

Pour continuer, il y a une classe de d rivation pour regrouper les m thodes appliqu es sur les comptes. Cette classe s'appelle compte et comporte les GET() et les SET(), ainsi que le constructeur et le AddSolde().

Probl me chiffre ap s la virgule :

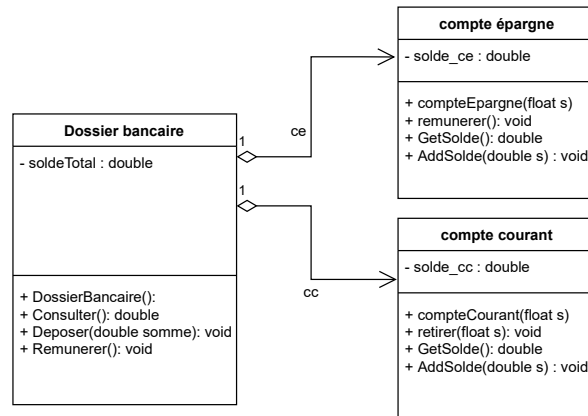
En utilisant le programme, nous nous sommes rendus compte que l'utilisation de la fonction Remunerer() donne de long chiffre   virgule. Or, nous souhaiterions savoir s'il existe des m thodes pour pr ciser le nombre de chiffres apr s la virgule dans un Double en JAVA.

Si nous avons besoin d'un nombre ainsi que de deux chiffres apr s la virgule, vraisemblablement pour du calcul financier, il faudrait travailler directement avec des int et consid rer ceux-ci comme une repr sentation en centimes. Dans ce cas, nous aurons des virgule fixe!

Les double et les float sont dits   "virgule flottante", c'est   dire que le nombre de chiffres avant et apr s la virgule n'est pas d fini.

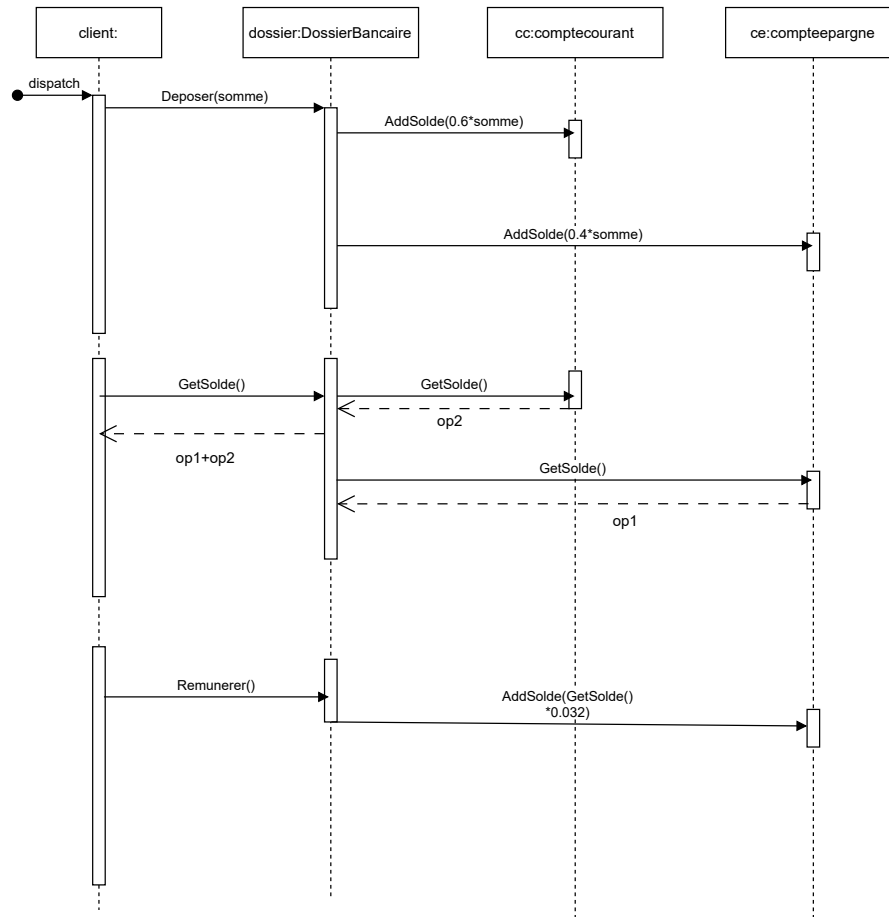
1.1 Diagramme UML du projet

Apr s le premier diagramme, nous nous sommes rendus compte que notre structure  tait inad quate, il est plus simple de travailler avec seulement trois classes comme ci-dessus.



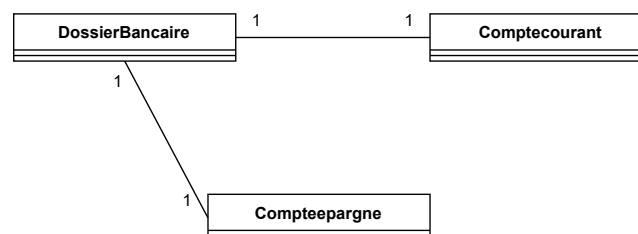
1.2 Diagramme UML de séquences

Voici notre diagramme des séquences suivant qui présente les différents appels de méthodes et leurs relations.

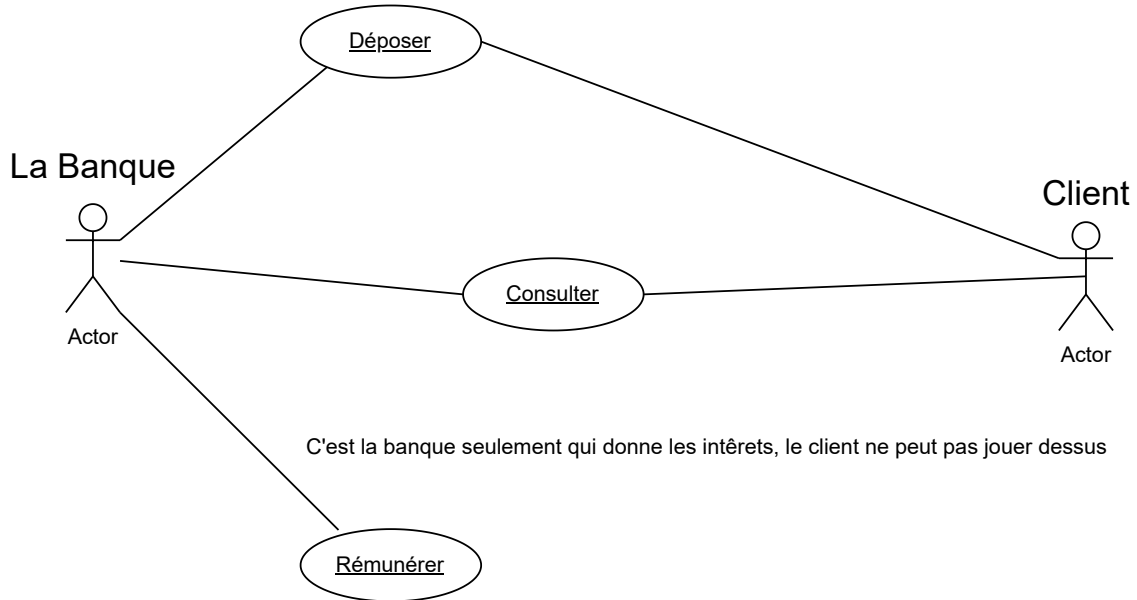


1.3 Diagramme d'objets et diagramme de cas d'utilisation

En plus des diagrammes des questions précédentes, nous pouvons réaliser un diagramme d'objets modélisant le programme.



De plus, nous avons décidé d'implémenter un dernier diagramme pour notre projet, le diagramme de cas d'utilisation. Effectivement, c'est un type de diagramme UML comportemental qui est utilisé pour analyser notre système. Il permet de visualiser les rôles du système et la façon dont ils interagissent avec le système "Dossier Bancaire".

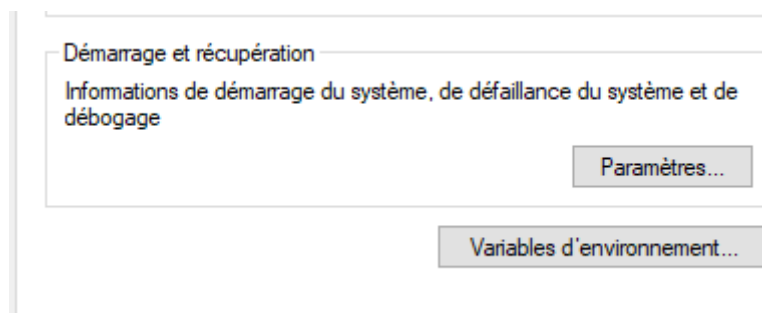


2 Code de départ

2.1 Exercice2

Nous allons maintenant voir comment compiler puis exécuter notre programme en ligne de commande sans passer par un IDE tel qu'Eclipse par exemple. Premièrement, nous avons repéré le compilateur afin de compiler du code java qui est : **JavaC**. Une fois cela fait, on remarque que nous ne pouvons pas l'utiliser si nous ne sommes pas dans le bon répertoire où il se trouve. Nous allons donc faire en sorte de pouvoir l'utiliser n'importe où en ajoutant sa localisation à la variable d'environnement \$Path.

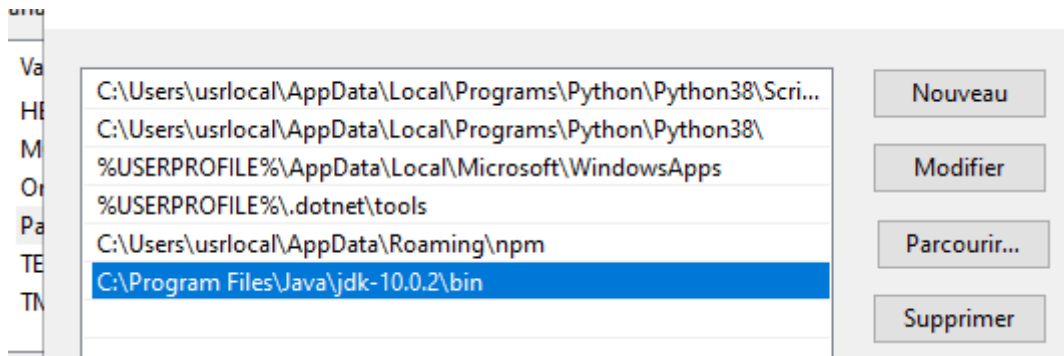
Pour se faire on se rend dans le panneau de configuration puis dans les paramètres avancés comme suit :



Nous allons ensuite dans les variables d'environnement où nous cherchons celle demandées :

OneDrive	C:\Users\usrlocal\OneDrive
Path	C:\Users\usrlocal\AppData\Local\Programs\Python\Python38\Scri...
TEMP	C:\Users\usrlocal\AppData\Local\Temp

Ensuite, nous ajoutons le chemin d'accès à javac pour pouvoir compiler dans n'importe quel répertoire .



Nous pouvons apercevoir une nouvelle ligne avec le chemin d'accès pour le compilateur javac. Avec cette démarche, cela nous permet de compiler et exécuter le programme en ligne de commande :

```

27 C:\Users\usrlocal\G-nie_logiciel\JavaProgSujet\src>javac myPackage/Main.java
28
29 C:\Users\usrlocal\G-nie_logiciel\JavaProgSujet\src>java myPackage/Main
30

```

En résumé, la variable environnement path de java doit contenir le lieu de tout les executables choisis.

Le but étant de réussir la compilation des tests, il reste des étapes à accomplir.

- Télécharger hamcrest-core, JUnit 4.
- Placer les fichiers .jar.
- Utiliser les commandes avec les bons chemins .
- Compiler et exécuter les tests.
- `javac -classpath ".../joint.jar":".../hamcrest.jar": Test/TestDossierBancaire.java`
- `java -classpath ".../joint.jar":".../hamcrest.jar": Test/TestDossierBancaire`

3 Développement: Exercice3

3.1 Question 1 et 2

Dans cette partie développement Java, nous devons créer un dépôt git dans le répertoire du projet. Pour ce faire en utilisant le logiciel Eclipse, nous utilisons les fonctions «Team», «Share project», «git», puis «use or create a repository in parent folder of project» ensuite «create repository», et pour finir «finish». Le répertoire de notre projet contient donc un fichier «.git/»

nom	modification	type	taille
.git	21/01/2023 00:44	Dossier de fichiers	
bin	20/01/2023 20:31	Dossier de fichiers	
src	02/12/2015 15:00	Dossier de fichiers	
.classpath	30/11/2015 22:15	Fichier CLASSPATH	1 Ko
.project	01/12/2015 06:54	Fichier PROJECT	1 Ko

Pour ajouter les fichiers de départ au dépôt git nous avons utilisé les commandes :

- git init
- 'git add .
- 'git commit -m "Commit initial"

```
commit 2fda5007554ad8863cc3f9d2544b583f8305d422 (HEAD -> master)
Author: maaxcode <74791725+maaxcode@users.noreply.github.com>
Date: Fri Jan 20 20:38:10 2023 +0100

    Commit initial
```


3.2 Question 3

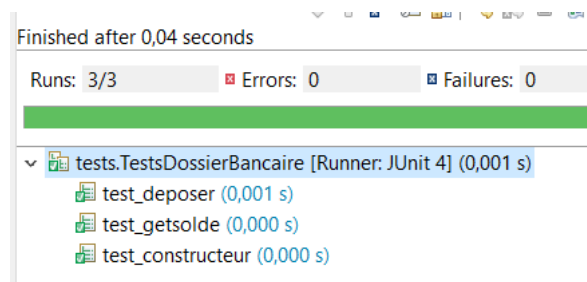
Pour cette partie nous avons mis en place des tests unitaires pour tester les méthodes du projet.

```

1 package tests;
2
3 import static org.junit.Assert.*;
4
5 //The Test annotation indicates that the public void method to which it is attached can be run as
6
7 public class TestsDossierBancaire {
8
9     @Test
10     public void test_constructeur()
11     {
12         DossierBancaire dossier=new DossierBancaire();
13         assertNotNull(dossier);
14     }
15
16     @Test
17     public void test_deposer()
18     {
19         DossierBancaire dossier=new DossierBancaire();
20         dossier.deposer(100);
21         assertEquals(100,dossier.get_solde(),0);
22     }
23
24     @Test
25     public void test_getsolde()
26     {
27         DossierBancaire dossier=new DossierBancaire();
28         assertEquals(0,dossier.get_solde(),0);
29     }
30 }

```

L'exécution de ces tests est valide comme le montre l'image suivante.



3.3 Question 4

Nous intégrons ces modifications à git et nous «taggons» cette version initiale avec : (V1.0)

```

MINGW64/c/Users/maxim/Downloads/genielog/JavaProgSujet
3 files changed, 36 insertions(+), 25 deletions(-)
create mode 100644 bin/tests/TestsDossierBancaire.class
delete mode 100644 src/tests/MyTest1.java
create mode 100644 src/tests/TestsDossierBancaire.java

Maxime@Lenovo-Legio-Y520 MINGW64 ~/Downloads/genielog/JavaProgSujet (master)
$ git tag V1.0

Maxime@Lenovo-Legio-Y520 MINGW64 ~/Downloads/genielog/JavaProgSujet (master)
$ git log
commit 1b875b96ddd924433bfbbbd8c7ddba744bed4f (HEAD -> master, tag: V1.0)
Author: maaxcode <74791725+maaxcode@users.noreply.github.com>
Date: Fri Jan 20 23:58:22 2023 +0100

    Classe_test_fusionné

commit 2fda5007554ad8863cc3f9d2544b583f8305d422
Author: maaxcode <74791725+maaxcode@users.noreply.github.com>
Date: Fri Jan 20 20:38:10 2023 +0100

    Commit initial

Maxime@Lenovo-Legio-Y520 MINGW64 ~/Downloads/genielog/JavaProgSujet (master)
$

```

3.4 Question 5

```

1 package myPackage;
2
3 public class DossierBancaire {
4     private Compte_courant cc;
5
6     //Constructeur
7     public DossierBancaire()
8     {
9         cc = new Compte_courant(0);
10    }
11
12    public void deposer(double value) {
13        cc.Addsolde(value);
14    }
15    public double get_solde() {return cc.getsolde();}
16    public void remunerer() {}
17
18 }
19
20

```

```

1 package myPackage;
2
3 public class Compte_courant {
4     private double _solde_cc;
5
6     public Compte_courant (double s)
7     {
8         _solde_cc = s;
9     }
10
11    public void Addsolde(double val)
12    {
13        _solde_cc+=val;
14    }
15
16    public double getsolde() {return _solde_cc;}
17
18 }
19
20
21
22
23
24

```

En ce qui concerne les nouveaux tests pour mettre la suite à jour :

```

35
36 }
37
38 @Test
39 public void deposer2()
40 {
41
42     Compte_courant cc = new Compte_courant(0);
43     cc.Addsolde(100);
44     assertEquals(100,cc.getsolde(),0);
45 }
46
47 @Test
48 public void constructeur_cc()
49 {
50
51     Compte_courant cc = new Compte_courant(20);
52     assertEquals(20,cc.getsolde(),0);
53
54 }
55
56
57 }
58

```

Pour finir, nous ajoutons les deux fichiers à l'index et nous exécutons le "commit".

3.5 Question 6

```

1 package myPackage;
2
3 public class Compte_epargne {
4
5     private double _solde_ce;
6
7     public Compte_epargne (double s)
8     {
9         _solde_ce = s;
10    }
11
12    public void Remunerer()
13    {
14        _solde_ce+=0.032*_solde_ce;
15    }
16
17    public void Addsolde(double val)
18    {
19        _solde_ce+=val;
20    }
21
22    public double getsolde() {return _solde_ce;}
23
24 }
25

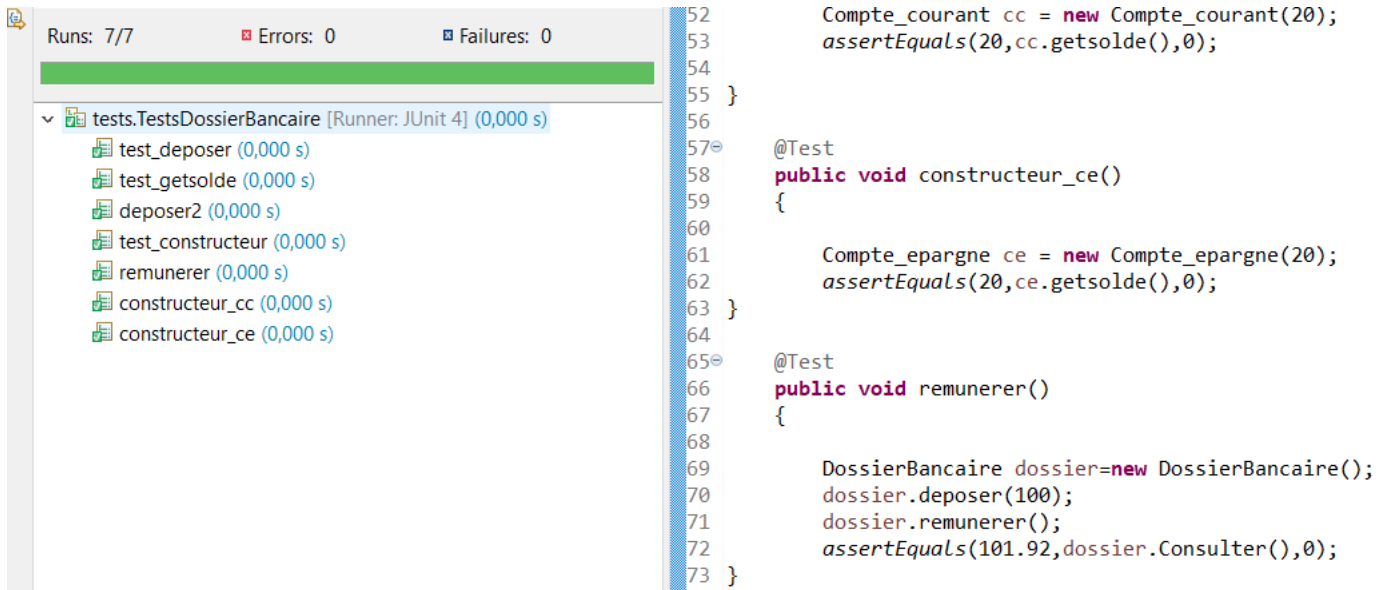
```

```

1 package myPackage;
2
3 public class DossierBancaire {
4
5     private Compte_courant cc;
6     private Compte_epargne ce;
7
8     //Constructeur
9     public DossierBancaire()
10    {
11        cc = new Compte_courant(0);
12        ce = new Compte_epargne(0);
13    }
14
15    public void deposer(double value)
16    {
17        cc.Addsolde(0.4*value);
18        ce.Addsolde(0.6*value);
19    }
20
21    public double Consulter() {
22        return cc.getsolde()+ce.getsolde();
23    }
24
25    public void remunerer() {
26        ce.Remunerer();
27    }
28
29 }
30

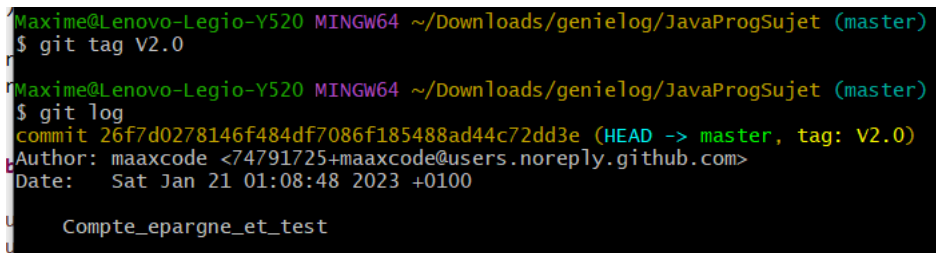
```

En ce qui concerne les nouveaux tests pour mettre la suite à jour :



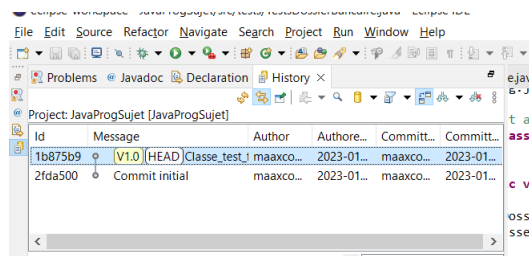
3.6 Question 7

Ici, nous mettons un tag V2.0 sur notre version.



3.7 Question 8

Dans cette partie, nous testons le retour la version antérieure 1.0. D'après les tests la rémunération est bien inactive.



3.8 Question 9

```
Maxime@Lenovo-Legio-Y520 MINGW64 ~/Downloads/genielog/JavaProgSujet ((V1.0))
$ git checkout master
Previous HEAD position was 1b875b9 Classe_test_fusionné
Switched to branch 'master'
D    src/tests/MyTest2.java
D    src/tests/MyTestSuite1.java
D    src/tests/MyTestSuite1Runner.java
```

A la fin de l'exercice, nous obtenons ce graphique :

Id	Message	Author	Authored Date	Committer	Comm
26f7d02	V2.0 new_dev HEAD Compte_epargne_et_test	maaxcode	2023-01-21 01:0...	maaxcode	2023-0...
f4f163b	Classe_compte_epargne	maaxcode	2023-01-21 00:4...	maaxcode	2023-0...
1b875b9	V1.0 Classe_test_fusionné	maaxcode	2023-01-20 23:5...	maaxcode	2023-0...
2fda500	Commit initial	maaxcode	2023-01-20 20:3...	maaxcode	2023-0...

4 Fusion : Exercice4

4.1 Question 1

Dans la classe «DossierBancaire», nous avons ajouté des commentaires pour enfin commiter.

4.2 Question 2

retour à une version précédente se fait via la commande :

- git checkout V2.0

Pour créer une nouvelle branche à partir de V2.0, il faut utiliser la commande git branch newdev puis git checkout newdev. Puis nous avons fait une amélioration de la structure du code en intégrant l'héritage afin de factoriser des éléments des classes compte courant et compte épargne.

```
Maxime@Lenovo-Legio-Y520 MINGW64 ~/Downloads/genielog/JavaProgSujet (new_dev)
$ git log
commit ce1e9e70ced93b74f9fe4037955f05cab0bbb973 (HEAD -> new_dev)
Author: maaxcode <74791725+maaxcode@users.noreply.github.com>
Date: Sat Jan 21 02:13:43 2023 +0100

    Implémentation de l'interface sur les comptes

commit 4f3ef9082f25cf7bab8677eb535431ee0837ba16
Author: maaxcode <74791725+maaxcode@users.noreply.github.com>
Date: Sat Jan 21 02:11:28 2023 +0100

    Ajout d'une interface pour structurer le code
```

4.3 Question 3

Pour se déplacer sur la branche master, en d'autres termes il faut déplacer la HEAD :

- git checkout master

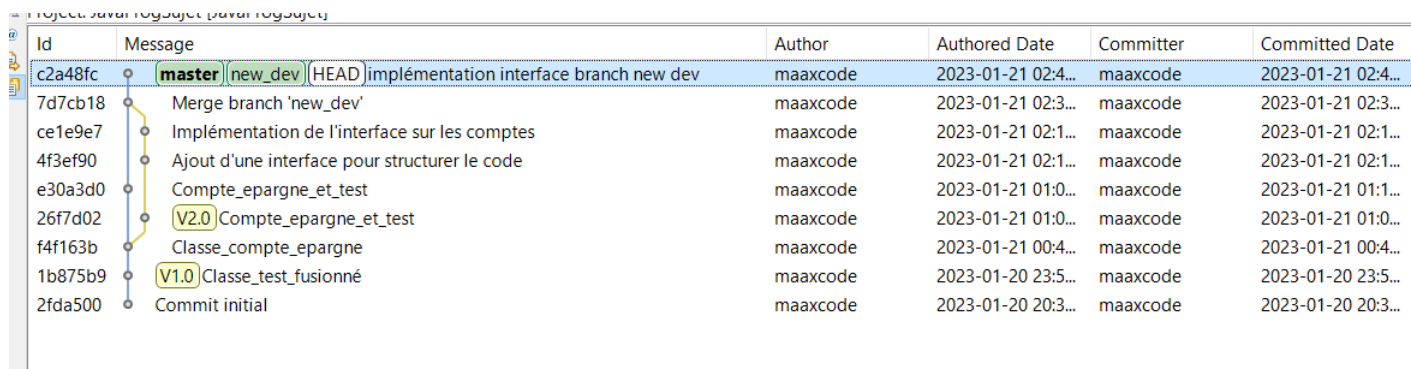
4.4 Question 4

Une fusion est utilisée pour incorporer les modifications apportées au nouveau développement dans la branche principale.

- git merge newdev master

4.5 Question 5

Pour finir, dans Eclipse, nous pouvons observer ce graphique représentant toutes les versions ainsi qu'une carte des branches.



Id	Message	Author	Authored Date	Committer	Committed Date
c2a48fc	implémentation interface branch new dev	maaxcode	2023-01-21 02:4...	maaxcode	2023-01-21 02:4...
7d7cb18	Merge branch 'new_dev'	maaxcode	2023-01-21 02:3...	maaxcode	2023-01-21 02:3...
ce1e9e7	Implémentation de l'interface sur les comptes	maaxcode	2023-01-21 02:1...	maaxcode	2023-01-21 02:1...
4f3ef90	Ajout d'une interface pour structurer le code	maaxcode	2023-01-21 02:1...	maaxcode	2023-01-21 02:1...
e30a3d0	Compte_epargne_et_test	maaxcode	2023-01-21 01:0...	maaxcode	2023-01-21 01:1...
26f7d02	V2.0 Compte_epargne_et_test	maaxcode	2023-01-21 01:0...	maaxcode	2023-01-21 01:0...
f4f163b	Classe_compte_epargne	maaxcode	2023-01-21 00:4...	maaxcode	2023-01-21 00:4...
1b875b9	V1.0 Classe_test_fusionné	maaxcode	2023-01-20 23:5...	maaxcode	2023-01-20 23:5...
2fda500	Commit initial	maaxcode	2023-01-20 20:3...	maaxcode	2023-01-20 20:3...

5 Tests : Exercice5

5.1 Question 1

Pour retirer de l'argent il faut modifier le Compte courant. Par contre, pour vérifier le solde il faut modifier DossierBancaire.

```

public void retirer(double value) throws Exception
{
    if(value > cc.getsolde()) {
        throw new Exception("Pas assez d'argent sur le compte bancaire pour retirer cette somme");
    }else{
        cc.retirer(value);
    }
}
}

```

```

1 package myPackage;
2
3 public class Compte_courant implements Compte{
4
5     private double _solde_cc;
6
7     public Compte_courant (double s)
8     {
9         _solde_cc = s;
10    }
11
12    public void Addsolde(double val)
13    {
14        _solde_cc+=val;
15    }
16
17    public double getsolde() {return _solde_cc;}
18
19    public void retirer(double value) {_solde_cc = _solde_cc - value;}
20
21
22
23

```

5.2 Question 2

Dans le cas ou suffisamment d'argent, le test fonctionne, dans le cas inverse cette exception est vérifiée par le test unitaire associée

Finished after 0,049 seconds

Runs: 9/9 Errors: 0 Failures: 1

tests.TestsDossierBancaire [Runner: JUnit 4] (0,001 s)

- test_retrait_fonctionnel (0,000 s)
- test_deposer (0,000 s)
- test_getsolde (0,000 s)
- deposer2 (0,000 s)
- test_constructeur (0,000 s)
- remunerer (0,000 s)
- Test_retrait_non_fonctionnel (0,001 s)**
- constructeur_cc (0,000 s)
- constructeur_cc (0,000 s)

Failure Trace

```

70 dossier.deposer(100);
71 dossier.remunerer();
72 assertEquals(101.92,dossier.Consulter(),0);
73 }
74
75 @Test
76 public void Test_retrait_fonctionnel() throws Exception
77 {
78     DossierBancaire dossier=new DossierBancaire();
79     dossier.deposer(150);
80     dossier.retirer(20);
81     assertEquals(130,dossier.Consulter(),0);
82 }
83
84 @Test
85 public void Test_retrait_non_fonctionnel()
86 {
87     DossierBancaire dossier=new DossierBancaire();
88     dossier.deposer(100); // on ne depose que 40 sur le compte courant
89     try {
90         dossier.retirer(41);
91     } catch (Exception e) {
92         fail("Echec du retrait car somme de retrait supérieure a celle du compte courant");
93     }
94 }
95

```

5.3 Question 3

```

if( e.getSource() == m_saisie_retirer )
{
    float retirier_value=Float.parseFloat(m_saisie_retirer.getText());
    try {
        m_dossier.retirer(retirier_value);
    } catch (Exception e1) {
        // TODO Auto-generated catch block
        e1.printStackTrace();
    }
    m_saisie_retirer.setText("");
}

```

6 Conclusion

Pour finir, nous avons une version différente de notre diagramme UML :

