École Polytechnique de Montréal
Département de génie informatique et génie logiciel
LOG1000 – Ingénierie logicielle

Travail pratique #4 :Introduction aux tests unitaires

Trimestre:Automne 2019

Equipier1 : Moussa Fofana 1955968
Equipier2 : Lompo Augustin 1927045

Equipe : 40

Présenté a : Charge de laboratoire

Polytechnique Montréal
17 Novembre 2019

4.1. Exécutez un test unitaire

Dans le code fourni « TP4.zip », il n'y a aucun test unitaire, ni projet pour exécuter les tests unitaires. Ajoutez un projet CppUnit et rédigez un test pour la fonction «Rabais::getRabais(Facture, int)». N'importe quel test peut faire l'affaire : l'objectif ici est d'avoir un projet CppUnit fonctionnel.
Les commandes pour tester doivent être ajoutées au Makefile existant.
Basez-vous sur les exemples et exercices faits en cours pour savoir comment monter un projet CppUnit.

```
#ifndef _RABAIS_TEST_
#define _RABAIS_TEST_


#include <cppunit/TestCase.h>
#include <cppunit/TestFixture.h>
#include <cppunit/extensions/HelperMacros.h>

#include "../../src/rabais.h"
#include "../../src/facture.h"
#include "../../src/client.h"


class Rabais_test: public CppUnit::TestFixture{


        CPPUNIT_TEST_SUITE(Rabais_test);
        CPPUNIT_TEST(rabaisEmploye);

        CPPUNIT_TEST_SUITE_END();

public:
     // Fonctions d'échafaudage
        void setUp();
        void tearDown();
    //les fonctions des tests
        void rabaisEmploye();

private:

        Rabais* rabais;

};
#endif
```

Figure 1: declaration de la classe rabais_test

```cpp
#include "rabais_test.h"

void Rabais_test::setUp(){
    this->rabais = new Rabais("data/clients.dat");

}
void Rabais_test::tearDown(){
    delete this->rabais;
}

// test d1=<{facture=220,ID= 25102},{rabais=0,15}>
void Rabais_test::rabaisEmploye(){
    Facture facture ;
    facture.ajouterItem(100.2);
    float r = this->rabais->getRabais(facture,25102);
    //CPPUNIT_ASSERT_EQUAL(0.15, rabais);
    CPPUNIT_ASSERT_DOUBLES_EQUAL((float) 0.15, r, FLT_EPSILON);
}
```

figure 2: implémentation de la classe Rabais_test

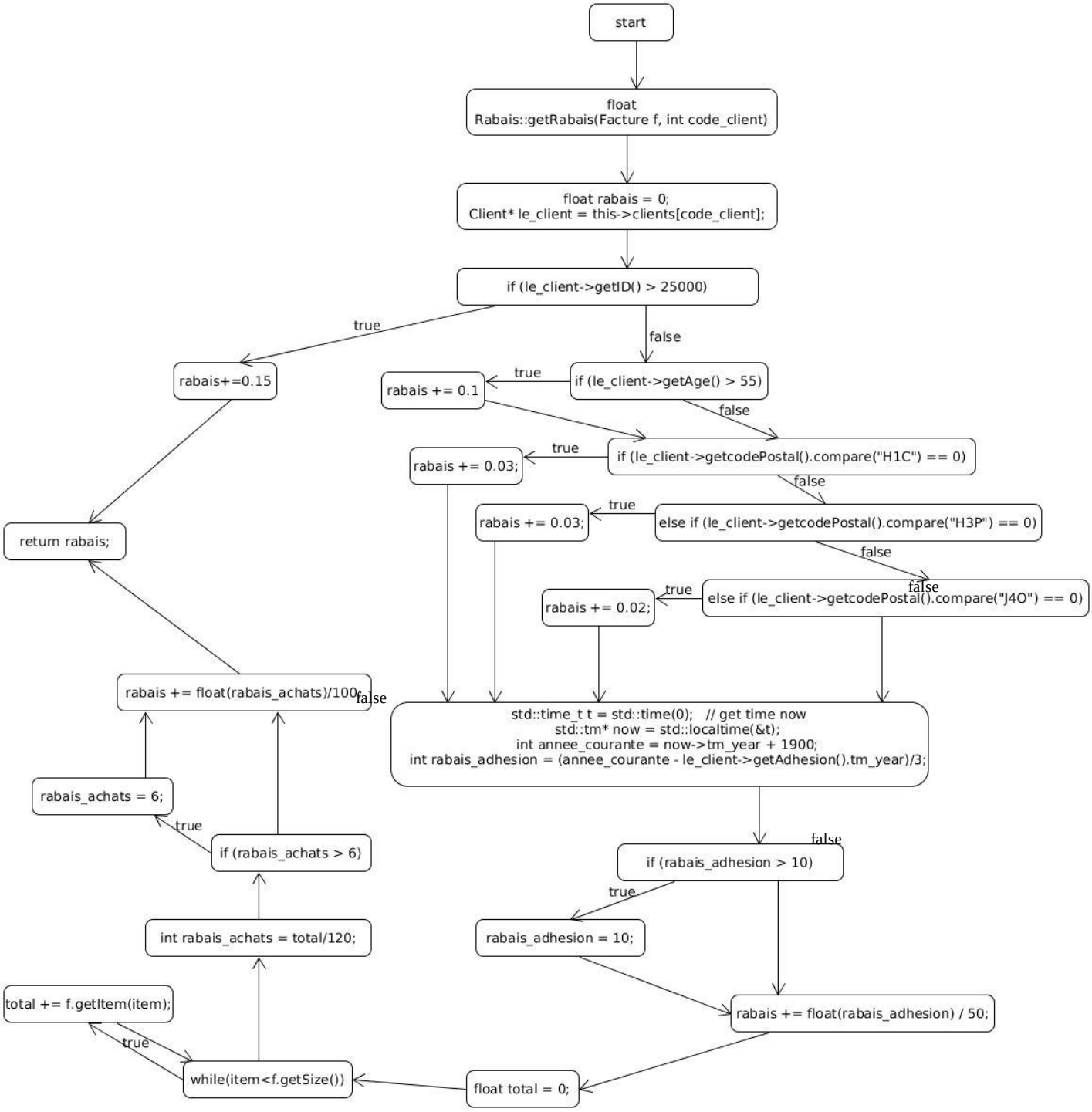
```
#================
# test get Rabais
#================
test: $(TESTS)/$(BINAIRE)/$(EXECTEST)
    ./$(TESTS)/$(BINAIRE)/$(EXECTEST)

$(TESTS)/$(BINAIRE)/$(EXECTEST): $(TESTS)/$(BINAIRE)/main.o $(TESTS)/$(BINAIRE)/rabais_test.o $(BINAIRE)/rabais.o $(BINAIRE)/client.o $(BINAIRE)/facture.o
    g++ -o $@ $^ -lcppunit

$(TESTS)/$(BINAIRE)/main.o: $(TESTS)/$(SOURCE)/main.cpp $(TESTS)/$(SOURCE)/rabais_test.h
    mkdir -p $(TESTS)/$(BINAIRE)
    g++ -o $@ -c $<

$(TESTS)/$(BINAIRE)/rabais_test.o: $(TESTS)/$(SOURCE)/rabais_test.cpp $(TESTS)/$(SOURCE)/rabais_test.h
    g++ -o $@ -c $<

# ===========
# Utilitaires
# ===========


# Enlève l'exécutable et les fichiers objets intermédiaires.
clean:
    rm -rf $(BINAIRE)/$(EXEC) $(BINAIRE)/*.o
    rm -rf $(TESTS)/$(BINAIRE)/*.o
    rm -rf $(TESTS)/$(BINAIRE)/$(EXEC)
```

Figure 3: capture de la modification sur le makefile

4.2. Rédigez le graphe de flot de contrôle du code

Rédigez le graphe de flot de contrôle (CFG, Control Flow Graph) pour le code de la fonction «Rabais::getRabais(Facture, int)». Placez le graphe dans le rapport.

Figure 4: Diagramme de graphe de flot *de* contrôle



4.3. <u>Rédigez les cas de tests pour la couverture des branches</u>

Rédigez, en utilisant le graphe de flot de contrôle, conceptuellement les cas de tests pour le code de la fonction «Rabais::getRabais(Facture, int)». Les cas de tests rédigés doivent assurer la couverture des branches de la fonction. Voici un exemple de cas de test rédigé de manière conceptuelle :

Nous avons au total sept branches qui sont:

- Employé : ID>25000
- age>55
- code postale=H1C
- code postale=H3P
- code postale=J4O
- année adhésion>10
- rabais additionnel>6

la complexité cyclomatique est 9 donc il nous faut au plus 9 tests pour couvrir toutes les branches alors nous allons effectuer 7 tests qui sont les suivants.
Pour couvrir toutes les branches nous avons ajouter deux autres client au fichier clients.dat

```
10456 Lamoureux Thomas 45 H1C 2016-10-25
14770 Lemay Marc 78 H2H 2010-01-30
15034 Thibault Danik 25 H3P 2019-01-15
12554 Michaud Charlotte 32 G4S 2015-08-08
25102 Tremblay Joseph 66 J4O 2008-12-13
14000 Etienne Konan 32 J4O 2005-02-03
15000 Van Hubert 32 T3K 1980-02-03
```

<u>Figure 4 :</u> modification du fichier clients

- **Rabais pour les employés.**

ici nous allons considérer que le client **Tramblay Joseph possède** des factures de 120$ et 100$, nous savons que son ID =25102>25000 donc est un employé donc son rabais est 15%
le test correspondant est:

d1=<{facture=220,ID= 25102},{rabais=0,15}>

- **Rabais pour les personnes de plus 55 ans**

Considérons le client **Lemay Marc** qui 78 ans et ayant une facture 100$ et 120$

d2 = <{facture=220, ID=14770},{rabais=0,17}>

- **Rabais pour un code postale H1C**

supposons un client **Lamoureux Thomas** ayant une facture de 100$ et 120$

d3 = <{facture=220, ID=10456},{rabais=0,06}>

- **Rabais pour un code postale H3P**

supposons un client **Thibault Danik** ayant une facture de 100$ et 120$

d4 = <{facture=220, ID=15034},{rabais=0,04}>

- **Rabais pour un code postale J4O**

supposons un client Etienne Konan ayant une facture de 100$ et 120$

d5 = <{facture=220, ID=14000},{rabais=0,11}>

- **Rabais adhésion>10**

  considérons le client 15000 Van Hubert 32 T3K 1980-02-03 ayant une facture 100$ et 120$ alors on a

  d6 = <{facture=220, ID=15000},{rabais=0,11}>

- **rabais additionnel>6**

  considérons le client **Michaud Charlotte** ayant une facture 1000$ alors on a

  d7 = <{facture=1000, ID=12554},{rabais=0,08}>

4.4. Codez les cas de tests dans le projet CppUnit

Codez les cas de tests conceptuels décrits précédemment dans le projet CppUnit. Assurez-vous d'identifier les cas de tests codés avec le même identifiant que les cas conceptuels, afin de faciliter la traçabilité entre vos cas conceptuels et les cas de tests codés.

Notez que vos tests unitaires ne doivent pas re-construire l'ensemble du logiciel. Ils doivent construire le strict minimum pour exécuter la fonction à tester. Vous n'avez pas à rédiger de «stubs» : vous pouvez utiliser les classes existantes et supposer qu'elles ont déjà été testées et qu'elles ne contiennent pas de défauts.

```
class Rabais_test: public CppUnit::TestFixture{


        CPPUNIT_TEST_SUITE(Rabais_test);
        CPPUNIT_TEST(rabaisEmploye);
        CPPUNIT_TEST(test_rabais_personne_55_ans);
        CPPUNIT_TEST(test_rabais_zone_H1C);
        CPPUNIT_TEST(test_rabais_zone_H3P);
        CPPUNIT_TEST(test_rabais_zone_J4O);
        CPPUNIT_TEST(test_rabais_annee_adhesion_sup10);
        CPPUNIT_TEST(test_rabais_achat_sup6);
        CPPUNIT_TEST_SUITE_END();

public:
     // Fonctions d'échafaudage
        void setUp();
        void tearDown();
    //les fonctions des tests
        void rabaisEmploye();
        void test_rabais_personne_55_ans();
        void test_rabais_zone_H1C();
        void test_rabais_zone_H3P();
        void test_rabais_zone_J4O();
        void test_rabais_annee_adhesion();
        void test_rabais_annee_adhesion_sup10();
        void test_rabais_achat_sup6();
```

Figure 5: les codes des cas de tests conceptuels

La sortie des tests dans le terminal

```
g++ -o test_getRabais/bin/main.o -c test_getRabais/src/main.cpp
g++ -o test_getRabais/bin/rabais_test.o -c test_getRabais/src/rabais_test.cpp
g++ -o test_getRabais/bin/rabais_test test_getRabais/bin/main.o test_getRabais/bin/rabais_test.o bin/rabais.o bin
/client.o bin/facture.o -lcppunit
./test_getRabais/bin/rabais_test
..F.F.F.F.F.F


!!!FAILURES!!!
Test Results:
Run:  7   Failures: 6   Errors: 0


1) test: Rabais_test::test_rabais_personne_55_ans (F) line: 26 test_getRabais/src/rabais_test.cpp
double equality assertion failed
- Expected: 0.140000000596046
- Actual  : 0.310000002384186
- Delta   : 1.19209289550781e-07


2) test: Rabais_test::test_rabais_zone_H1C (F) line: 35 test_getRabais/src/rabais_test.cpp
double equality assertion failed
- Expected: 0.0599999986588955
- Actual  : 0.240000009536743
- Delta   : 1.19209289550781e-07


3) test: Rabais_test::test_rabais_zone_H3P (F) line: 43 test_getRabais/src/rabais_test.cpp
double equality assertion failed
- Expected: 0.0399999991059303
- Actual  : 0.240000009536743
- Delta   : 1.19209289550781e-07


4) test: Rabais_test::test_rabais_zone_J4O (F) line: 52 test_getRabais/src/rabais_test.cpp
double equality assertion failed
- Expected: 0.109999999403954
- Actual  : 0.230000004172325
- Delta   : 1.19209289550781e-07


5) test: Rabais_test::test_rabais_annee_adhesion_sup10 (F) line: 61 test_getRabais/src/rabais_test.cpp
double equality assertion failed
- Expected: 0.109999999403954
- Actual  : 0.230000004172325
- Delta   : 1.19209289550781e-07


6) test: Rabais_test::test_rabais_achat_sup6 (F) line: 69 test_getRabais/src/rabais_test.cpp
double equality assertion failed
- Expected: 0.0900000035762787
- Actual  : 0.230000004172325
- Delta   : 1.19209289550781e-07
```

Figure 6: sortie du terminal après avoir codé les test conceptuels

4.5. Trouvez un défaut

Défauts

Dans la méthode Rabais::lireFichier, on obtient l'année d'adhésion par le calcul
: annee_string -1900;
Et dans la méthode Rabais::getRabais , l'annee_courante est obtenu en utilisant le calcul :now-
→tm_year + 1900 ce qui donne 2019 ;

Donc en voulant trouver le rabais_adhésion on aura toujours un rabais maximum >10 , Par exemple si annee_string est 2000 on aura l'année d'adhésion qui sera 2000-1900=100;
en outre, pour obtenir rabais_adhésion on fera : (2019-100)/3=639,67 ce qui engendrera un rabais_adhésion toujours supérieur a 10.
De plus le pourcentage de rabais_adhésion est toujours multiplié par 2 car on fait le calcul : rabais+=float(rabais_adhésion)/50 . En somme lorsque une personne qui n'est pas employée est entrée dans le système elle reçoit toujours un rabais d'adhésion de 20%.

Exigence non respectée

L'exigence SRS05 qui exige un rabais additionnel de 4% pour la zone HC1C n'est pas respectée car dans le code c'est plutôt un rabais de 3% qui est appliquée.
Aussi l'exigence SRS07 n'est pas respectée car le rabais basé sur la date adhésion peut aller jusque a 20%.

Solutions apportées au code

Pour remédier au problème lie au non respect de l'exigence SRS07 et au défaut trouve plus haut, il ne faut pas soustraire 1900 de la variable annee_string(donc on aura exactement la vrai date d'admission de la personne); aussi au lieu de fixer la limite de la variable rabais_adhésion a 10 il faudrait plutôt la fixer a 5.

Ensuite pour ce qui est de du non respect de l'exigence SRS05 il faut juste changer les 3% appliqué dans le code par les 4% qui devrait être appliqués.

```
[fomou@l3818-18 Code (master)] $ make test
g++ -o test_getRabais/bin/rabais_test.o -c test_getRabais/src/rabais_test.cpp
g++ -o test_getRabais/bin/rabais_test test_getRabais/bin/main.o test_getRabais/bin/rabais_test.o bin/rabais.o bin
/client.o bin/facture.o -lcppunit
./test_getRabais/bin/rabais_test
.......


OK (7 tests)
```

Figure 7: capture d'écran de la correction du code avec tous nos tests qui passent

4.6 Rétroaction

Nous avons en tout travaillé 10 heures-personnes , Oui l'effort demande pour ce travail pratique était très adéquat.