

Les instructions de la société BibTel

La société BibTel a consigné ses exigences dans le document intitulé « *Cahier des charges BibTel du projet "ToutAvis"* » (CDC) qui vous est remis au moment de la séance. Lisez-le attentivement.

Travail à réaliser

Lorsqu'on aborde le développement d'un produit, qu'il soit logiciel ou non, il est impératif d'en déterminer à l'avance les modalités de recette. Il s'agit d'établir comment on pourra, le moment venu, mesurer le degré de conformité du produit livré à la demande client¹.

L'objectif de ce BE est donc de préparer la validation de votre futur produit en élaborant les éléments décrits dans la suite. Cette validation se fera exclusivement à partir de l'API² (**Application Programming Interface**) du produit et donc sans utiliser l'IHM fournie. Cette dernière n'est en effet utile que pour montrer le fonctionnement en conditions réelles du produit après que celui-ci ait été validé. La raison de cela est que la procédure de recette doit s'exécuter de façon *automatique* et *reproductible* sans faire appel à l'interprétation de l'opérateur. Elle sera développée en amont du développement du produit pour permettre d'en valider l'intégration au fil de l'eau. Cette procédure sera basée sur des jeux de test modulaires de façon à pouvoir en automatiser l'enchaînement.

Le CDC Bibtel vous renseigne également sur les composants logiciels qui vous sont fournis (NB : Les modalités de son rapatriement dans votre environnement de développement Eclipse vous sont fournies sur Moodle).

Le livrable attendu contiendra les éléments suivants :

1. Les **fiches de scénario**³ pour validation des deux méthodes `addItemFilm()` et `reviewItemFilm()` correspondant à l'API décrite dans l'interface `ISocialNetwork`. Ces scénarios seront décrits sur les *fiches de validation* mises à votre disposition sur Moodle. À titre d'exemple, une fiche de validation de la fonction `addMember()` vous est fournie.
2. Le **code Java** correspondant à vos fiches. Ce code devra s'interfacer avec le code d'enchaînement de tests qui vous est fourni, et qui accumule les résultats des modules de test exécutés et affiche un compte-rendu général du type : « Nombre de tests passés, nombre de tests en échec ». Toujours à titre d'exemple, le code de test de la fonction `addMember()` correspondant à la fiche évoquée ci-dessus est également fourni. Efforcez-vous de produire un code modulaire en favorisant l'indépendance entre modules. De manière générale, il vous appartient dès à présent de respecter au mieux les recommandations fournies dans le « *Guide de bonnes pratiques en génie logiciel* » mis à votre disposition.
3. Une copie du **compte-rendu général d'exécution de votre code de test**. Comme `SocialNetwork` n'est pour l'instant qu'une coquille vide, votre test devrait détecter et afficher de nombreuses erreurs. C'est cette trace d'exécution que vous livrez.

Les livrables sont à déposer sur Moodle, sous forme d'une archive (.gz) unique par équipe, si possible avant la prochaine séance (BE3). Ils serviront de supports aux futures phases de

- 1 Nota : cette démarche reste valable si l'on est soi-même le client du produit qu'on développe ou s'il s'agit d'une demande interne à l'entreprise.
- 2 Cette API est spécifiée par l'interface `ISocialNetwork` décrite dans le cahier des charges Bibtel.
- 3 Dans la rédaction de ces fiches de test, il faut bien choisir la granularité de celles-ci (le titre qu'on leur donne peut aider à cela) de manière à garder leur nombre dans des limites raisonnables (une douzaine par exemple).

validation, mais pour cela, ils devront alors être complétés par tous les cas de test qui, bien que non exigés dans le présent livrable, seront utiles et nécessaires à la validation finale du produit livré.

Méthode de travail

Adoptez une démarche incrémentale :

1. installez l'infrastructure logicielle fournie par le client dans Eclipse, et vérifiez que le test fourni s'exécute correctement (et détecte des erreurs dans `addMember()` puisque `SocialNetwork` est une coquille vide pour l'instant).
2. Faites le scénario de test pour les cas anormaux de `addItemFilm()`, puis implémentez le code Java correspondant, et testez.
3. Faites de même pour les cas nominaux de `addItemFilm()`.
4. Attaquez-vous enfin à `reviewItemFilm()`.

Complément sur la problématique du test

À lire... et à méditer

Une problématique centrale du test provient du choix du point de départ de l'opération : quelles sont les fonctions de l'API par lesquelles on va commencer la validation ? A priori, il convient de suspecter de dysfonctionnement l'ensemble de ces fonctions, et cette suspicion s'étend même au code de test qu'on est en train de développer ! On se retrouve donc à utiliser un outillage dont la correction est incertaine pour tester la conformité d'un produit à un cahier des charges. On mesure la difficulté de l'exercice et dès lors la nécessité d'élaborer une stratégie sérieuse avant de se lancer.

Quelques principes permettent de mener à bien cette périlleuse mission :

1. Stratégie de test : commencer par les fonctions les plus simples et dont le couplage avec les autres fonctions du système est minimum, par exemple celles relatives aux membres ;
2. Développer le test selon une démarche *incrémentale* qui consiste à assurer un pas avant de passer au suivant.
3. S'efforcer d'examiner le plus exhaustivement possible l'espace d'états que le système peut prendre et garder à l'esprit qu'un acteur humain a, *a priori*, un comportement non prédictible.
4. S'il ne faut pas oublier de tester des cas « anormaux » de fonctionnement (typiquement, les cas de levées d'exception), il ne faut pas négliger le test des *cas dits « nominaux »* !

Exemples de fiches de validation et du code correspondant

Vous trouverez sur Moodle le dossier complet de validation à compléter. Il est constitué :

- d'une fiche de renseignement ;
- d'un bilan de test ;
- et de fiches décrivant les scénarios de test pour valider individuellement, autant que faire se peut, chaque fonction de l'API.

Pour vous aider à concevoir ces fiches, on vous livre ci-après, à titre d'exemple, deux fiches destinées à la validation de la fonction `addMember()`. N'hésitez pas à vous en inspirer pour le reste.

À suivre, vous trouverez également le code Java correspondant à l'implémentation de ces deux fiches.

Fiche de test N° 1

Objectif du test : éprouver la méthode addMember sur les cas d'anomalie

Vérification de la levée de l'exception BadEntryException lors de l'utilisation de addMember « avec des paramètres d'entrées incorrects ».

Pour rappel : Les 5 cas de levée de l'exception BadEntryException par la méthode addMember prévus dans l'API sont :

- si le pseudo n'est pas instancié ou a moins de 1 caractère autre que des espaces,
- si le password n'est pas instancié ou a moins de 4 caractères, autres que des « leading or trailing blanks »,
- si le profil n'est pas instancié.

Description (scénario) :

- Instanciation d'un Social Network
- 1.1 Utilisation de addMember avec en paramètre un pseudo non instancié.
- 1.2 Utilisation de addMember avec en paramètre un pseudo ne contenant que des espaces.
- 1.3 Utilisation de addMember avec en paramètre un password non instancié.
- 1.4 Utilisation de addMember avec en paramètre un password de moins de 4 caractères, autres que des « leading or trailing blanks ».
- 1.5 Utilisation de addMember avec en paramètre un profil non instancié.


Résultats attendus :

Dans les 5 utilisations de addMember, l'exception BadEntryException doit être levée.

De plus, après levée de l'exception BadEntryException lors de l'utilisation de addMember, le nombre de membres doit rester identique au nombre de membres avant utilisation de addMember.

Résultats observés :

Conclusion :

 <p>IMT Atlantique Bretagne-Pays de la Loire École Mines-Télécom</p>	<p>FIP INF112 – GL & Gestion de projet informatique</p> <p>Livret fil rouge</p> <p>BE2</p>	<p>Année scolaire</p> <p>2019 – 2020</p> <p>Page : 4/4</p>
<p>BE2 : préparation de la validation du prototype à réaliser</p> <p>V20.2</p>		

Fiche de test N° 2

Objectif du test : éprouver la méthode addMember sur les cas de fonctionnement standard

- But principal : vérification de l'ajout de membres après utilisation de addMember « avec des paramètres d'entrées corrects »
- But secondaire : vérification de la levée de l'exception MemberAlreadyExistsException par la méthode addMember pour tous les cas prévus dans l'API.

Pour rappel, le cas de levée de l'exception MemberAlreadyExistsException par la méthode addMember prévu dans l'API est :

- si un membre de même pseudo est déjà présent dans le SocialNetwork (même pseudo : indifférent à la casse et aux *leading* et *trailing blanks*)

Description (scénario) :

- Instanciation d'un Social Network
- 2.1 Utilisations de addMember pour l'ajout de plusieurs membres « avec paramètres d'entrées corrects »
- 2.2 Utilisation de addMember avec en paramètre un pseudo identique au pseudo du premier membre « correct » ajouté.
- 2.3 Utilisation de addMember avec en paramètre un pseudo identique au pseudo du dernier membre « correct » ajouté.
- 2.4 Utilisation de addMember avec en paramètre le pseudo d'un membre existant (mais avec casse différente).
- 2.5 Utilisation de addMember avec en paramètre le pseudo d'un membre existant (mais avec des leading/trailing blanks).

Résultats attendus :

L'ajout de membres « avec paramètres d'entrées corrects » ne doit pas lever d'exception, le nombre de membres après ces ajouts doit avoir augmenté en conséquence.

Dans les 4 utilisations suivantes de addMember, l'exception MemberAlreadyExistsException doit être levée.

De plus après levée de l'exception MemberAlreadyExistsException lors de l'utilisation de addMember, le nombre de membres doit rester identique au nombre de membres avant utilisation de addMember.

Résultats observés :

Conclusion :

oct. 30, 18 18:36

AddMemberTest.java

Page 1/4

```

package tests;

import opinion.ISocialNetwork;
import opinion.SocialNetwork;

import exceptions.BadEntryException;
import exceptions.MemberAlreadyExistsException;
import exceptions.NotTestReportException;

/**
 * Tests for the SocialNetwork.<i>addMember()</i> method.
 *
 * @author B. Prou, E. Cousin, GO
 * @version V2.0 - April 2018
 */
public class AddMemberTest {

    /**
     * Check that trying to add this new member (login, pwd, profile) raises a
     * BadEntry exception and does not change content of the
     * <i>ISocialNetwork</i>. If OK, the method just returns 0. If not OK,
     * displays an error message and returns 1.
     *
     * @param sn
     *      - the <i>ISocialNetwork</i>
     * @param login
     *      - new member's login
     * @param pwd
     *      - new member's password
     * @param profile
     *      - new member's profile
     * @param testId
     *      - the test ID that will prefix any error message displayed by
     *        this method
     * @param errorMessage
     *      - the error message that will be displayed if no exception is
     *        thrown when adding this member
     * @return 0 if the test is OK, 1 if not
     */
    private static int addMemberBadEntryTest(ISocialNetwork sn, String login,
        String pwd, String profile, String testId, String errorMessage) {

        int nbMembers = sn.nbMembers(); // Number of members when starting to
        // run this method

        try {
            sn.addMember(login, pwd, profile); // Try to add this member
            // Reaching this point means that no exception was thrown by
            // addMember()
            System.out.println("Err " + testId + ":" + errorMessage); // display
                                                                    // the
                                                                    // error
                                                                    // message

            return 1; // and return the "error" value
        } catch (BadEntryException e) { // BadEntry exception was thrown by
            // addMember() : this is a good start!
            // Let's now check if 'sn' was not
            // impacted
            if (sn.nbMembers() != nbMembers) { // The number of members has
                // changed : this is an error
                // case.

                System.out
                    .println("Err "
                        + testId
                        + " : BadEntry was thrown but the number of members was changed"); // Display
                                                                    // a
                                                                    // specific
                                                                    // error
                                                                    // message

                return 1; // return "error" value
            } else
                // The number of members hasn't changed, which is considered a
                // good indicator that 'sn' was not modified
                return 0; // return success value : everything seems OK, nothing
                // to display
        } catch (Exception e) { // An exception was thrown by addMember(), but
            // it was not the expected exception BadEntry
            System.out.println("Err " + testId + " : unexpected exception. "
                + e); // Display a specific error message
            e.printStackTrace(); // Display contextual info about what happened
            return 1; // return error value
        }
    }

    /**
     * Check that trying to add this new member (login, pwd, profile) raises an
     * AlreadyExists exception and does not change content of the
     * <i>ISocialNetwork</i>. If OK, the method just returns 0. If not OK,
     * displays an error message and returns 1.
     *
     * @param sn
     *      - the <i>ISocialNetwork</i>

```

oct. 30, 18 18:36

AddMemberTest.java

Page 2/4

```

* @param login
*         - new member's login
* @param pwd
*         - new member's password
* @param profile
*         - new member's profile
* @param testId
*         - the test ID that will prefix any error message displayed by
*           this method
* @param errorMessage
*         - the error message that will be displayed if no exception is
*           thrown when adding this member
* @return 0 if the test is OK, 1 if not
*/
private static int addMemberAlreadyExistsTest(ISocialNetwork sn,
String login, String pwd, String profile, String testId,
String errorMessage) {
    int nbMembers = sn.nbMembers(); // Number of members when starting to
    // process this method
    try {
        sn.addMember(login, pwd, profile); // Try to add this member
        // Reaching this point means that no exception was thrown by
        // addMember()
        System.out.println("Err " + testId + ":" + errorMessage); // display
                                                                    // the
                                                                    // error
                                                                    // message
        return 1; // and return the "error" value
    } catch (MemberAlreadyExistsException e) { // AlreadyExists exception was
        // thrown by addMember() :
        // this is a good start!
        // Let's now check if 'sn'
        // was not impacted
        if (sn.nbMembers() != nbMembers) {
            System.out
                .println("Err "
                    + testId
                    + ": MemberAlreadyExists was thrown, but the number of members was changed"); // Display
                                                                    // a
                                                                    // specific
                                                                    // error
                                                                    // message

            return 1; // and return the "error" value
        } else
            return 0; // return success value : everything is OK, nothing to
            // display
    } catch (Exception e) { // An exception was thrown by addMember(), but
        // it was not the expected exception
        // AlreadyExists
        System.out.println("Err " + testId + ": unexpected exception. "
            + e); // Display a specific error message
        e.printStackTrace(); // Display contextual info about what happened
        return 1; // return error value
    }
}

/**
 * Check that this new member (login, pwd, profile) can be (and <i>is</i>)
 * added to the <i>ISocialNetwork</i>.</br> If OK, the method just returns 0
 * : the new member was added.</br> If not OK, an error message is displayed
 * and 1 is returned ; the new member was not correctly added.
 */
* @param sn
*         - the <i>ISocialNetwork</i>
* @param login
*         - new member's login
* @param pwd
*         - new member's password
* @param profile
*         - new member's profile
* @param testId
*         - the test ID that will prefix any error message displayed by
*           this method
* @return 0 if the test is OK, 1 if not
*/
private static int addMemberOKTest(ISocialNetwork sn, String login,
String pwd, String profile, String testId) {
    int nbMembers = sn.nbMembers(); // Number of members when starting to
    // process this method
    try {
        sn.addMember(login, pwd, profile); // Try to add this member
        // No exception was thrown. That's a good start !
        if (sn.nbMembers() != nbMembers + 1) { // But the number of members
            // hasn't changed
            // accordingly
            System.out.println("Err " + testId
                + ": the number of members (" + nbMembers
                + ") was not incremented"); // Error message displayed
            return 1; // return error code
        } else
            return 0; // return success code : everything is OK, nothing to
            // display
    } catch (Exception e) { // An exception was thrown by addMember() : this

```

```

        // is an error case
        System.out
            .println("Err " + testId + ":unexpected exception " + e); // Error
                                                                    // message
                                                                    // displayed
        e.printStackTrace(); // Display contextual info about what happened
        return 1; // return error code
    }
}

/**
 * <i>addMember()</i> main test :
 * <ul>
 * <li>check if members can be added</li>
 * <li>check if incorrect parameters cause addMember() to throw BadEntry
 * exception</li>
 * <li>check if adding already registered members cause addMember() to throw
 * AlreadyExists exception</li>
 * </ul>
 *
 * @return a summary of the performed tests
 */
public static TestReport test(){

    ISocialNetwork sn = new SocialNetwork();

    int nbBooks = sn.nbBooks(); // number of books in 'sn' (should be 0
                                // here)
    int nbFilms = sn.nbFilms(); // number of films in 'sn' (should be 0
                                // here)

    int nbTests = 0; // total number of performed tests
    int nbErrors = 0; // total number of failed tests

    System.out.println("Testing addMember()");

    // <=> test nÂ°1

    // check if incorrect parameters cause addMember() to throw BadEntry
    // exception

    nbTests++;
    nbErrors += addMemberBadEntryTest(sn, null, "qsdfgh", "", "1.1",
        "addMember() doesn't reject null logins");
    nbTests++;
    nbErrors += addMemberBadEntryTest(
        sn,
        " ",
        "qsdfgh",
        "",
        "1.2",
        "addMember() doesn't reject logins that don't contain at least one character other than space");
    nbTests++;
    nbErrors += addMemberBadEntryTest(sn, "B", null, "", "1.3",
        "addMember() doesn't reject null passwords");
    nbTests++;
    nbErrors += addMemberBadEntryTest(
        sn,
        "B",
        " qwd ",
        "",
        "1.4",
        "addMember() doesn't reject passwords that don't contain at least 4 characters (not taking into account leading or trailing blanks)");
    nbTests++;
    nbErrors += addMemberBadEntryTest(sn, "BBBB", "bbbb", null, "1.5",
        "addMember() doesn't reject null profiles");

    // <=> test nÂ°2

    // populate 'sn' with 3 members

    nbTests++;
    nbErrors += addMemberOKTest(sn, "Paul", "paul", "lecteur impulsif",
        "2.1a");
    nbTests++;
    nbErrors += addMemberOKTest(sn, "Antoine", "antoine",
        "grand amoureux de la litt  rature", "2.1b");
    nbTests++;
    nbErrors += addMemberOKTest(sn, "Alice", "alice",
        "passionn  e de bande dessin  e", "2.1c");

    // try to add already registered members

    nbTests++;
    nbErrors += addMemberAlreadyExistsTest(sn, new String("Paul"),
        "abcdefghij", "", "2.2",
        "The login of the first member was accepted as login for a new member");
    nbTests++;
    nbErrors += addMemberAlreadyExistsTest(sn, new String("Alice"),
        "abcdefghij", "", "2.3",
        "The login of the last member was accepted as login for a new member");
    nbTests++;
    nbErrors += addMemberAlreadyExistsTest(

```

```

        sn,
        new String("anToine"),
        "abcdefghij",
        "",
        "2.4",
        "An already registered login, but with different case, was accepted as login for a new member" );
nbTests++;
nbErrors += addMemberAlreadyExistsTest (
    sn,
    new String(" Antoine "),
    "abcdefghij",
    "",
    "2.5",
    "An already registered login, surrounded by leading/trailing blanks, was accepted as login for a new member" );
nbTests++;
nbErrors += addMemberAlreadyExistsTest (
    sn,
    "An" + "toi" + "ne",
    "abcdefghij",
    "",
    "2.6",
    "A String concatenation building an already registered login was accepted as login for a new member" );

nbTests++;
// check that 'sn' was not modified
if (nbFilms != sn.nbfilms()) {
    System.out
        .println("Error : the number of films was unexpectedly changed by addMember()");
    nbErrors++;
}
nbTests++;
if (nbBooks != sn.nbBooks()) {
    System.out
        .println("Error : the number of books was unexpectedly changed by addMember()");
    nbErrors++;
}

// Display final state of 'sn'
System.out.println("Final state of the social network : " + sn);

// Print a summary of the tests and return test results
try{
    TestReport tr = new TestReport(nbTests, nbErrors);
    System.out.println("AddMemberTest:" + tr);
    return tr;
}
catch (NotTestReportException e){ //This shouldn't happen
    System.out.println("Unexpected error in AddMemberTest test code - Can't return valuable test results");
    return null;
}
}

/**
 * Launches test()
 * @param args not used
 */
public static void main(String[] args) {
    test();
}
}

```