David ANATON

CORBA Messenger

Manuel développeur

1. Table des matières

2. Historique des modifications

Name	*Date (dd/mm/yy)*	*Reason For Changes*	*Version*

3. Introduction

Ce manuel est là pour expliquer l'approche utilisée pour développer l'application Messenger permettant de mettre en réseau plusieurs clients pour que ces derniers puissent échanger des messages textuels.

4. Travail effectué

4.1. Server Side

Ayant choisi de travailler avec Corba, le développement a commencé avec la rédaction d'un fichier .idl.

Ce dernier spécifie l'ensemble des fonctions accessibles à travers le réseau. La compilation de ce dernier générera un ensemble de classes qui permettront aux clients d'appeler ces fonctions sur le serveur.

4.1.1. Messenger.idl

Contient les prototypes des méthodes accessibles sur le serveur depuis le client.

Compilé en utilisant `idlj -fall Messenger.idl`

4.1.2. Dossier MessengerApp

Celui-ci est généré automatiquement par compilation du fichier IDL.

4.1.3. MessengerImpl.java

Implémentation des méthodes spécifiées dans le fichier IDL.

4.1.4. StartServer.java

Se connecte à l'ORB et met en place un serveur qui attendra les connexions de clients sur un objet java de type MessengerImpl.

4.2. Client Side

4.2.1. Dossier MessengerApp

Celui-ci est généré automatiquement par compilation du fichier IDL dans le serveur. Il a été importé pour avoir accès aux différents objets de l'ORB.

4.2.2. StartClient

Connexion à l'ORB. Mise en place d'une boucle qui demande les commandes à effectuer aux clients.

4.3. Tests Unitaires

4.3.1. Tests

Dans un souci de simplicité, les tests ont été effectués sur un projet de test.

Vous trouverez donc un projet supplémentaire s'appelant TestsUnitaires. Il contient une calculatrice. On appuie sur chaque touche l'une après l'autre et le résultat s'affiche dans l'invite de commande.

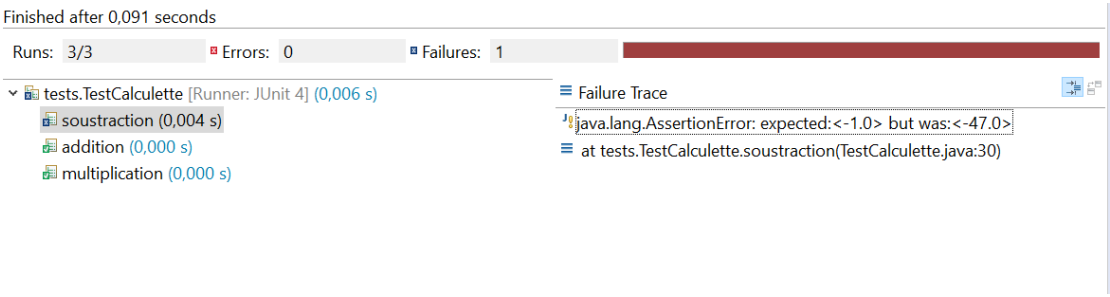
Les tests fonctionnent de la façon suivante : on crée un package nommé « test » qui les contiendra tous. Pour chaque classe nécessitant des tests, on crée un nouveau JUnit Test Case.

Dans cette classe, nous définissons ensuite les comportements attendus de la classe. Par exemple, Lorsque l'on appuie successivement sur les boutons « 4 + 5 = », on attend 9 comme résultat.

Main.java | Calculette.java | *TestCalculette.java

```
1  package tests;
2
3  import static org.junit.Assert.*;
8
9  public class TestCalculette {
10
11     @Test
12     public void addition() {
13         Calculette c = new Calculette();
14         c.presserBouton("4");
15         c.presserBouton("+");
16         c.presserBouton("5");
17         c.presserBouton("=");
18         double resultat = c.getResultat();
19         assertEquals(9, resultat, 0.000000001);
20     }
21
22
23 }
24
```

De cette manière, au lancement de l'application, les portions de code ne donnant pas le résultat escompté sont signalées. Dans notre cas, la soustraction n'est pas bien codée puisqu'elle renvoie 47 au lieu de 1.



4.3.2. Couverture de code

Un outil supplémentaire, appelé EclEmma et basé sur Jacoco, permet de s'assurer que ces test sont bien présents sur l'intégralité du code.

Après son installation depuis le marketplace, on peut l'utiliser pour voir les portions de codes non testées. On peut ainsi voir que la méthode « diviser ( ) » est la seule qui n'a pas son test implémenté.

```
46    private void additionner(){
47        this.resultat += Double.parseDouble(this.value);
48        this.value = "";
49    }
50
51    private void soustraire(){
52        this.value += "1";
53        this.resultat -= Double.parseDouble(this.value);
54        this.value = "";
55    }
56
57    private void diviser(){
58        this.resultat /= Double.parseDouble(this.value);
59        this.value = "";
60    }
61
62    private void multiplier(){
63        this.resultat *= Double.parseDouble(this.value);
64        this.value = "";
65    }
```

5. Améliorations envisagées

5.1. Interface graphique

5.2. Possibilité de créer des chambres de discussion différentes

5.3. Récupération des messages en temps réel

5.3.1. Récupération à intervalles réguliers

Mise en place d'un thread chez chaque client qui lance la commande #messages toutes les 5 secondes.

5.3.2. Récupération en temps réel

Chaque client devient un mini-serveur qui peut recevoir les messages au fur et à mesure que le serveur les lui envoie.

6. Manuel Utilisateur

6.1. ORBD

Choisir un port.

Aller en ligne de commande (Windows + R, puis taper « `cmd` »). Naviguer jusqu'au dossier du projet et lancer la commande `start orbd - ORBInitialPort <numéro de port>`

6.2. Serveur

Naviguer ensuite dans le sous-projet du server et lancer le StartServer en lui fournissant les paramètres suivants `-ORBInitialPort <numéro de port> -ORBInitialHost localhost`

6.1. Client

Naviguer dans le sous-projet du client et lancer le StartClient en lui passant les paramètres suivants -
`ORBInitialPort <numéro de port>-ORBInitialHost <addresse du serveur>`

Le client est alors invité à entrer son nom, puis le nom de son correspondant. Il a ensuite accès à l'interface de communication.

Il peut alors entrer un texte à envoyer à son correspondant ou plusieurs commandes :

- `#clients` : Retourne la liste de tous les clients connectés
- `#exit` : Déconnecte son client de l'application
- `#help` : Affiche la liste des commandes disponibles
- `#pseudo` : Changer de pseudo
- `#to` : Changer de destinataire
- Chaine de caractères vide : télécharge la liste des messages adressés au client

STAY IN TOUCH !