

Maison des ligues

Tests unitaires et fonctionnels

Rapport de tests

Pour simplifier, vous savez qu'il existe deux grands types de tests :

- les tests unitaires : qui permettent de tester une méthode isolée du reste
- les tests fonctionnels : qui permettent de tester une fonctionnalité de l'application (qui peut mettre en jeu plusieurs méthodes)

Pour ces deux types de tests, vous pouvez travailler en réalisant les tests par vous-même avec des jeux d'essais préparés, ou vous pouvez utiliser les outils de tests mis à disposition par l'IDE. La seconde solution est bien sûr préférable.

De plus, les tests peuvent être réalisés à deux niveaux différents :

- au niveau de l'application : afin de tester le code de l'application
- au niveau de la base de données : afin de tester le code directement inséré dans la base

Tests dans la base de données

Méthodologie

Voyons tout de suite un exemple concret de test dans la base de données :

- Ouvrez SQL Developer
- Menu Outils/Test d'unité/Sélectionner le référentiel en cours...
- Dans la liste de connexions, choisir la connexion mdl que vous avez créée, puis ok
- Tapez le mot de passe correspondant à mdl et ok
- Lors de la première création de tests :
 - Outils/Test d'unité/Créer Mettre à jour le référentiel
 - Normalement la première fois vous n'avez pas les permissions : cliquez sur ok pour voir les permissions qui devront être appliquées. Vous êtes invité à vous loguer sous l'utilisateur sys : tapez le mot de passe et ok. Les ordres grant apparaissent : cliquez sur oui. Plusieurs fenêtres vont défiler dans le même genre, où vous devrez faire soit ok, soit oui pour exécuter les ordres sql proposés.
 - Après une petite attente, vous obtenez le message "Référentiel créé", cliquez sur ok
- Création d'un test unitaire :
 - Affichage/Test d'unité : un nouvel onglet "Test d'unité" est apparu
 - Click droit sur Tests, sélectionnez "Créer un test"
 - La fenêtre d'assistance montre à gauche la progression des étapes de création d'un test
 - A droite, sélectionnez la connexion mdl
 - Dans la zone centrale, développez Packages, FONCTIONS DIVERSES et sélectionnez DUREEVACATION (cette fonction est dans la correction, mais si vous ne l'avez pas, vous pouvez sélectionner une autre fonction que vous avez créée). Cliquez sur suivant
 - Un nom est donné au test. Laissez sélectionné "Créer avec une seule implémentation dummy", cliquez sur suivant
 - Dans la partie "Démarrer le processus", ne faites rien et cliquez sur suivant

- C'est la partie paramètres qui est la plus intéressante. Pour les paramètres en entrée, vous pouvez directement saisir des valeurs dans la colonne Entrée. Idem pour les paramètres en sortie, dans la colonne Résultat. Si vous avez sélectionné la fonction DUREEVACATION, elle n'a pas de paramètre en entrée, en revanche elle est censée retourner la durée de vacation qui est de 90. Donc saisissez 90 dans la colonne Résultat du paramètre. Cliquez sur suivant
- Pas de validation à spécifier : cliquez sur suivant
- Pas de détachement : cliquez sur suivant
- Vous obtenez un récapitulatif : cliquez sur Fin
- Utilisation du test :
 - A gauche, développez Tests et sélectionnez la fonction de test qui vient d'être créé
 - Dans la fenêtre centrale, cliquez sur la flèche verte ("Exécuter le test").
 - Si le test se passe bien, vous obtenez plusieurs lignes avec à chaque fois, en début de ligne, une coche verte. La dernière ligne précise, dans la colonne Message, le résultat attendu et le résultat obtenu (qui doivent être les mêmes). Dans la colonne Statut, vous devriez obtenir les messages SUCCESS.

Tests à effectuer pour cette mission

Maintenant que vous connaissez la méthode, le but est de créer des tests sur toutes les procédures que vous aurez créées dans les différentes tâches (en particulier la tâche 2 où il est clairement demandé de créer des procédures stockées pour réaliser le travail).

Tests dans l'application

Visual Studio offre un outil puissant qui permet de gérer les tests sur l'application.

Tests unitaires

Les tests unitaires sont faits pour tester une classe indépendamment du reste de l'application, plus précisément des méthodes publiques qui retournent une information (donc, de type fonction). Le reste du code peut éventuellement être testé suite à des appels dans des méthodes "testables" : par exemple, une méthode privée ou qui ne retourne rien peut tout de même être testée si elle est appelée par une autre méthode qui, elle, est publique et retourne une information.

Dans le code de la correction, je n'ai pas d'exemple de méthodes qui s'adapte à un test unitaire. Cependant, vous en avez peut-être dans votre code. Et pour montrer le principe sous VS, on va créer une méthode. Dans la classe FrmPrincipale.cs (ou une autre classe publique), ajoutez la méthode suivante :

```
public int calcul(int a, int b)
{
    return a+b ;
}
```

Vous l'avez compris, c'est juste pour voir le principe.

Voici maintenant comment créer simplement un test unitaire sous VS :

- Menu Test/Nouveau test
- Dans la fenêtre, sélectionnez "Assistant test unitaire" puis ok
- Donnez un nom (par exemple EssaiTest1) puis Créer
- Dans la nouvelle fenêtre, développez le projet MaisonDesLigues (en cliquant sur +), puis développez encore MaisonDesLigues, puis FrmPrincipale, et sélectionnez Calcul (vous avez compris qu'il serait tout à fait possible de sélectionner plusieurs méthodes), puis cliquez sur ok
- Au bout d'un moment, vous obtenez un nouvel onglet avec le fichier FrmPrincipalTest.cs
- Observez le contenu : vous retrouvez en particulier la méthode calculTest qui est déjà écrite. Il suffit de

modifier le contenu des variables a et b (pour le moment à 0) ainsi que le contenu de expected (qui est la valeur attendue en sortie). Remplissez a et b avec respectivement 3 et 5, et remplissez expected avec 8.

- Si en dernière ligne de la méthode, vous avez un Assert.Inconclusive, mettez cette ligne en commentaire.
- Pensez à enregistrer, puis cliquez sur la flèche verte de gauche, qui s'appelle "Exécuter les tests dans le contexte actuel"
- Au bout d'un moment, dans la zone du bas, vous obtenez le résultat du test : normalement "réussite", avec une coche verte.
- Pour contrôler que le test peut aussi échouer, modifiez le contenu de expected (par exemple mettez 7), et relancez le test. Cette fois vous obtenez un échec. Si vous regardez le message d'erreur, il est précisé qu'il était attendu 7 alors que la valeur actuelle est 8.

Maintenant que vous avez compris le principe sous VS, vous pouvez essayer de faire d'autres tests unitaires, mais globalement cette application ne s'y prête pas vraiment. Il va falloir plutôt réaliser des tests fonctionnels.

Tests fonctionnels

Les tests fonctionnels au niveau de l'interface ont pour but de contrôler que des manipulations donnent le résultat attendu et ne provoquent pas d'erreurs. Vous allez alors dire "mais à quoi cela sert d'enregistrer des manipulations puisqu'on peut les faire directement pour tester". En fait, une fois que ces manipulations sont enregistrées, vous pourrez les "rejouer" pour tester à nouveau cette fonctionnalité plus tard ? A quoi cela sert-il de retester plus tard une fonctionnalité ? Il arrive que quand on ajoute du code, cela ajoute des erreurs dans du code existant : on parle de "régression". Le fait de rejouer un ancien test va permettre de contrôler que l'ancienne fonctionnalité déjà testée, marche toujours correctement, malgré l'ajout d'un nouveau code.

Concrètement, voyons un exemple de création d'un tel test :

- Menu Test/Nouveau test
- Dans la fenêtre, sélectionnez "Test codé de l'interface utilisateur" puis ok
- Donnez un nom au test et cliquez sur Créer
- Dans la nouvelle fenêtre, sélectionnez le premier choix "Enregistrer les actions, modifier le mappage ou ajouter des assertions", puis ok
- Un générateur de test codé de l'interface s'ouvre, avec des messages explicatifs
- Lancez l'application et connectez-vous
- Cliquez sur le rond rouge ("Démarrer l'enregistrement"). A partir de maintenant, toutes vos manipulations vont être enregistrées (vous remarquerez les messages qui indiquent les manipulations)
- Faites une manipulation qui correspond à un test d'une fonctionnalité que vous venez d'ajouter (par exemple, allez dans "Ajout atelier" et ajoutez un atelier puis enregistrez).
- Cliquez maintenant sur "Suspendre l'enregistrement"
- Vous pouvez à tout moment afficher les étapes enregistrées (bouton à côté du rond rouge)
- Quittez l'application
- Il suffit maintenant de générer le code en cliquant sur le bouton "Générer le code" : un nom de méthode est demandé, donnez un nom et cliquez sur "Ajouter et générer"
- Vous remarquez un onglet qui s'est ajouté, contenant le test fonctionnel (le code généré est en réalité stocké dans la partie design de la classe UIMap)
- Si vous avez ajouté un atelier lors du test, allez le supprimer dans SQL Developer (pensez d'abord à supprimer les thèmes que vous avez mis dans l'atelier ainsi que les vacances)
- On va maintenant rejouer le film :
 - Lancez l'application et connectez-vous, ainsi vous êtes positionné à l'endroit où vous aviez commencé l'enregistrement.
 - Dans VS, sélectionnez l'onglet du test et cliquez sur la flèche verte de gauche "Exécutez les tests dans le contexte actuel"

- puis revenez à la fenêtre d'exécution de l'appli (qui est donc en cours d'exécution) : vous allez remarquer que les manipulations que vous aviez enregistrées se font automatiquement, comme si quelqu'un manipulait à votre place.
- Une fois les manipulations terminées, vous pouvez quitter l'application et revenir à VS.
- Remarquez que dans la partie basse, vous avez le message Réussite avec la coche verte : le test fonctionnel s'est bien déroulé.

En fonction du code écrit dans la tâche 4, vous pouvez créer des tests fonctionnels qui vont contrôler chaque manipulation.

Rapport de tests

Un rapport de tests permet de recenser les tests effectués, les résultats attendus, les résultats obtenus et le bilan. Prenons par exemple le test sur la fonction Calcul. Le rapport pourrait se présenter ainsi :

Type de test	Test unitaire
Object concerné	fonction CALCUL
Fonctionnalité	Reçoit en paramètre 2 entiers et retourne la somme.
Conditions initiales	a=3 ; b=5
Résultat attendu	8
Résultat obtenu	8
Etat du test	Réussi
Bilan du test	La fonction est opérationnelle pour 2 entiers strictement positifs.

Si en revanche le test échoue, c'est intéressant dans le rapport de préciser les erreurs corrigées avant d'obtenir un test qui réussit. On peut bien sûr imaginer plusieurs tests pour la même fonction.

Voici un exemple de rapport pour un test fonctionnel :

Type de test	Test fonctionnel
Object concerné	fenêtre FrmPrincipal
Fonctionnalité	Ajout d'un atelier
Conditions initiales	Saisie d'un atelier et d'un thème avec demande d'enregistrement
Résultat attendu	Saisies et enregistrement effectués
Résultat obtenu	Saisies et enregistrement effectués
Etat du test	Réussi
Bilan du test	L'ajout d'un atelier avec un thème fonctionne correctement.

Pour l'ensemble des tests que vous allez effectuer, pensez donc à créer un rapport correspondant.

Tâche 6 : Documentation technique

Pas de génération possible de documentation sous sqldeveloper pour le code Oracle.

Par contre tout est documenté.

Sous Visual studio, il existe un moyen standardisé de commentaire. Placez 3 slashes avant une méthode et Visual Studio vous créera un cartouche avec quelques balises XML vous guidant dans votre documentation de code :

```
/// <summary>
/// Méthode événementielle qui va permettre de créer un nouveau thème pour l'atelier en
/// cours de création
/// La méthode va créer un nouveau libellé et une nouvelle zone de saisie.
/// Il va aussi redimensionner et repositionner les panels.
/// </summary>
/// <param name="sender">controle déclencheur</param>
/// <param name="e">eventargs</param>
```

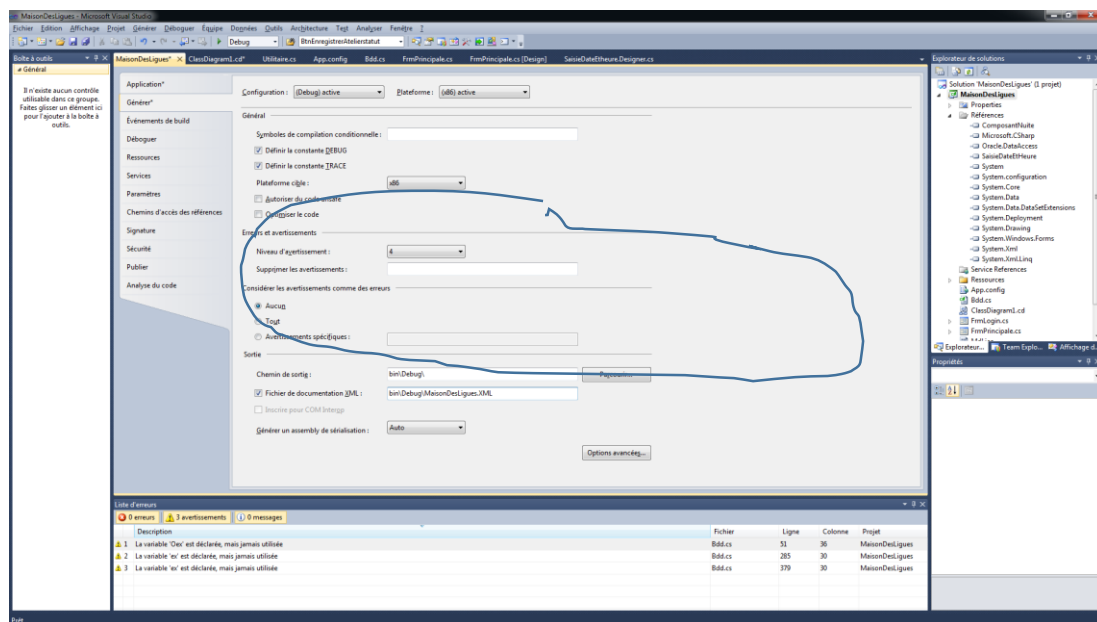
Il existe d'autres balises que vous pouvez ajouter. Un lien intéressant :

<http://msdn.microsoft.com/fr-fr/library/vstudio/b2s063f7%28v=vs.100%29.aspx>

Génération de documentation :

Aller dans le menu projet/propriétés du projet/onglet générer

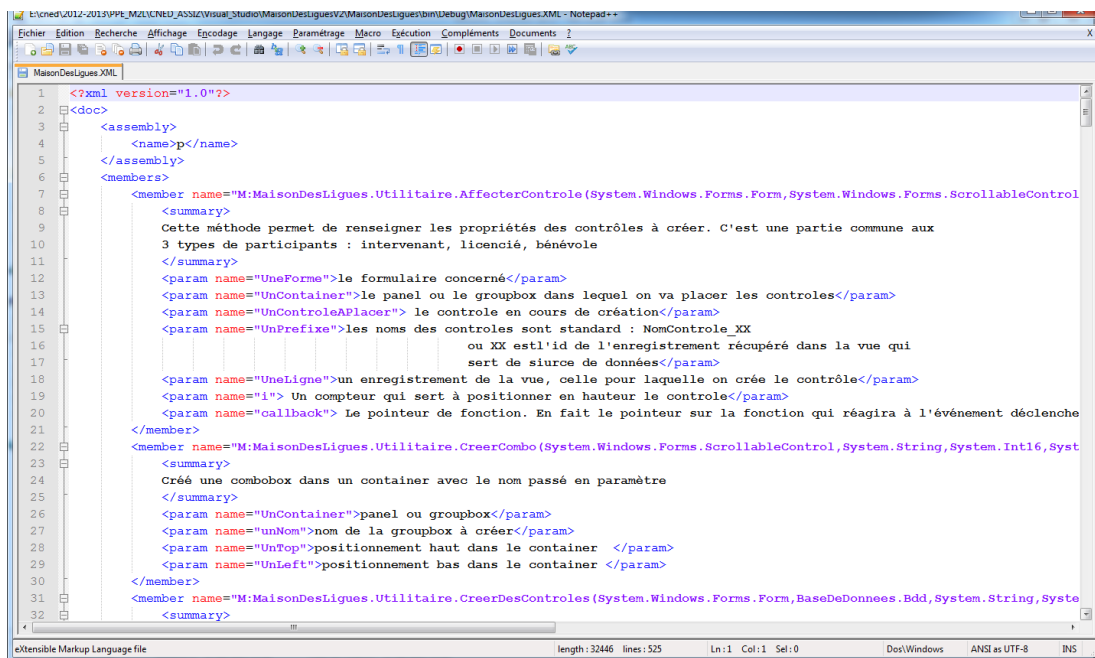
Cochez la case Fichier de documentation XML et renseignez le chemin et le nom du fichier. le fichier sera généré à chaque compilation C'est pour cette raison qu'il vaut mieux faire cette opération à la fin, quand vous avez terminé votre projet.



Remarque : si il vous manque des commentaires.... Vous verrez des avertissements dans la fenêtre liste d'erreurs :

Liste d'erreurs			
0 erreurs 14 avertissements 0 messages			
Description	Fichier	Ligne	Colonne
14 Le paramètre 'UnIdAtelier' ne possède pas de balise param correspondante dans le commentaire XML pour 'BaseDeDonnees.Bdd.MettreAJourVacations (System.Collections.ObjectModel.Collection<string>, short)' (contrairement à d'autres paramètres)	Bdd.cs	31	30
4 Le paramètre 'UnContainer' ne possède pas de balise param correspondante dans le commentaire XML pour 'MaisonDesLigues.Utilitaire.CompteChecked	Utilitaire.c	285	30

Extrait du fichier généré :



```
<?xml version="1.0"?>
<doc>
  <assembly>
    <name>p</name>
  </assembly>
  <members>
    <member name="M:MaisonDesLigues.Utilitaire.AffecterControl(System.Windows.Forms.Form,System.Windows.Forms.ScrollableControl
    <summary>
      Cette méthode permet de renseigner les propriétés des contrôles à créer. C'est une partie commune aux
      3 types de participants : intervenant, licencié, bénévole
    </summary>
    <param name="UneForme">le formulaire concerné</param>
    <param name="UnContainer">le panel ou le groupbox dans lequel on va placer les controles</param>
    <param name="UnControlAPlacer"> le controle en cours de création</param>
    <param name="UnPrefixe">les noms des controles sont standard : NomControl_XX
      ou XX est l'id de l'enregistrement récupéré dans la vue qui
      sert de source de données</param>
    <param name="UneLigne">un enregistrement de la vue, celle pour laquelle on crée le contrôle</param>
    <param name="i"> Un compteur qui sert à positionner en hauteur le controle</param>
    <param name="callback"> Le pointeur de fonction. En fait le pointeur sur la fonction qui réagira à l'événement déclenche
  </member>
    <member name="M:MaisonDesLigues.Utilitaire.CreerCombo(System.Windows.Forms.ScrollableControl,System.String,System.Int16,Syste
    <summary>
      Créé une combobox dans un container avec le nom passé en paramètre
    </summary>
    <param name="UnContainer">panel ou groupbox</param>
    <param name="unNom">nom de la groupbox à créer</param>
    <param name="UnTop">positionnement haut dans le container </param>
    <param name="UnLeft">positionnement bas dans le container </param>
  </member>
    <member name="M:MaisonDesLigues.Utilitaire.CreerDesControles(System.Windows.Forms.Form,BaseDeDonnees.Bdd,System.String,Syste
    <summary>
```

Missions envisagées :

Dans la fenêtre de modification d'ajouts, mettre les contrôles en lecture seule et un bouton en face qui fait passer le contrôle en lecture seule et qui fait les mises à jour seulement des vacations modifiées.

Conclusion

Si vous désirez tester la correction proposée, voici la procédure pour sa mise en place :

- Ouvrir le projet visual studio Maison des ligues
- Faire apparaitre le composant SaisieDateEtHeure dans la boite à outil

Executer sous PLSQL les scripts suivants :

- sous l'utilisateur system :
 - creer_user_mdIM1.sql
- Sous l'utilisateur mdl et dans l'ordre :
 - creer_objetsM1.SQL
 - creer_package_M1.sql
 - creer_triggerM1.sql
 - creer_enregistrementsM1.SQL
- sous l'utilisateur system :
 - creer_user_employemdlM1.sql