Tests d'acceptation avec FitNesse

Bruno Mermet

2010

–

ébauche

Plan

- Présentation générale de FitNesse
- Langage de marquage : référence
- Types de table *Slim* : référence

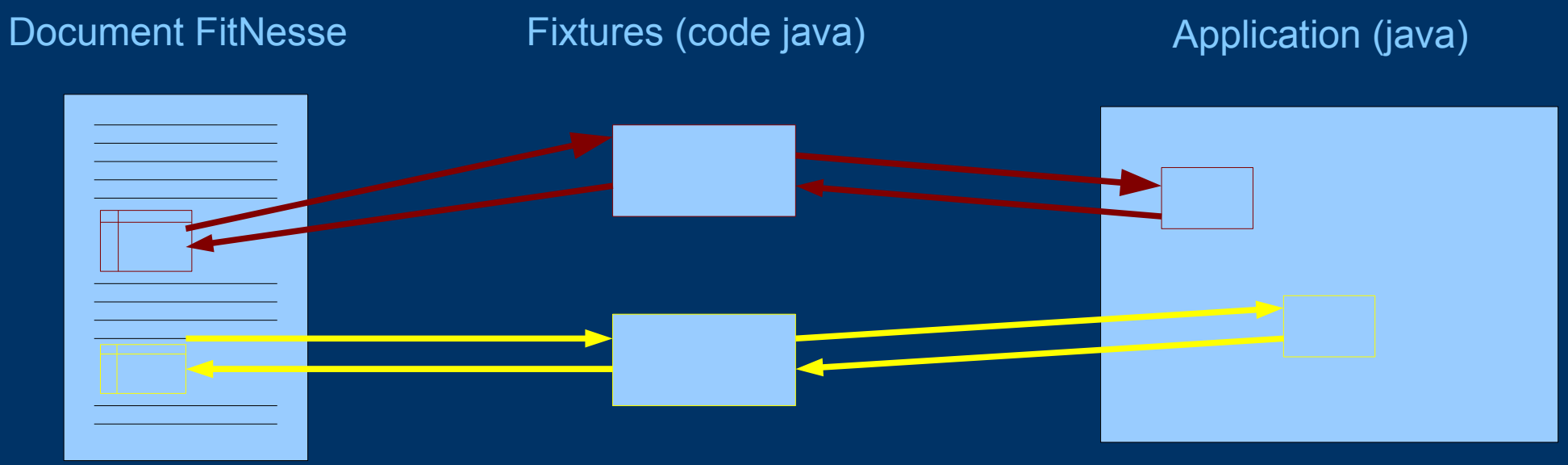
Présentation générale de FitNesse

Tests d'acceptation / tests unitaires

- Tests unitaires
  - Orientés « petite unité de code »
  - Compréhension réservée aux développeurs
- Tests d'acceptation (Tests de recette)
  - Orientés « fonctionnalité »
  - Compréhension accessible au client

FitNesse en bref

- Présentation générale
  - Un outil type wiki permettant de spécifier des tests d'acceptation au milieu de texte informel
  - Serveur autonome, donc facile à installer
- Fonctionnement général



Installation de FitNesse

- Récupérer l'archive (fitnesse.jar) et mettre le jar là où l'on veut installer le logiciel

  http://fitnesse.org/FrontPage.FitNesseDevelopment.DownLoad

- Exécuter une première fois l'archive
  - java -jar fitnesse.jar
- Exécuter une deuxième fois l'archive
  - Soit : java -jar fitnesse.jar (port 80)
  - Soit : java -jar fitnesse.jar -pXXXX (port XXXX)
- Lancer son navigateur et se connecter sur la bonne machine et le bon port (http://localhost ou http://localhost:XXXX)

Exemple de page FitNesse : code

```
!1 PremierExemple : un jeu de tests avec FitNesse
!define TEST_SYSTEM {slim}
!contents -R2 -g -p -f -h
!path C:\Users\bruno\Documents\NetBeansProjects\ArbresPPO\build\test\classes
!path C:\Users\bruno\Documents\NetBeansProjects\ArbresPPO\build\classes

!2 Importation
|import|
|correction5|

!2 Un premier jeu de tests
|creation arbre|
|valeur | parent | position | taille? | nbFeuilles? | hauteur? |
| 10    |  0     | gauche   |   1     |    1        |    0     |
| 20 | 1 | gauche | 2 | 1 | 1|
| 30 | 1 | droite | 3 | 2 | 1|
| 40 | 2 | gauche | 4 | 2 | 2|
| 50 | 4 | droite | 5 | 2 | 3|

On va bien voir ce que donne ce tableau...
```

Titre niveau 1

Utilisation des tests slim et non fit

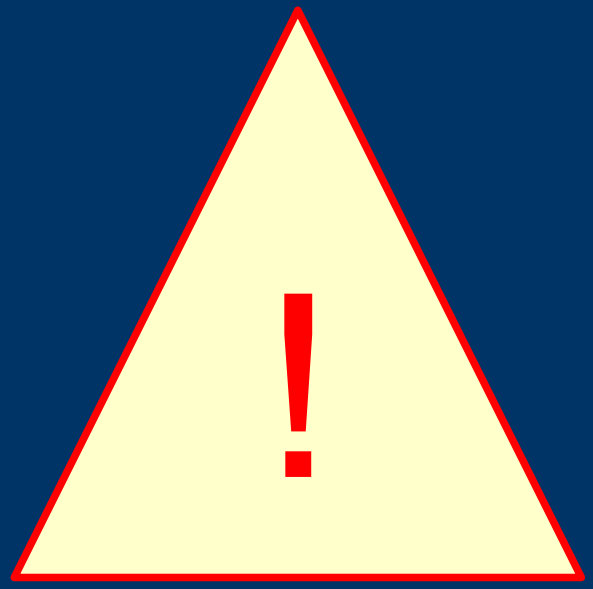
Table des matières

Titre niveau 2

Table de type « import » pour définir les paquetages importés

Définition du CLASSPATH avec
- les « fixtures »
- l'application

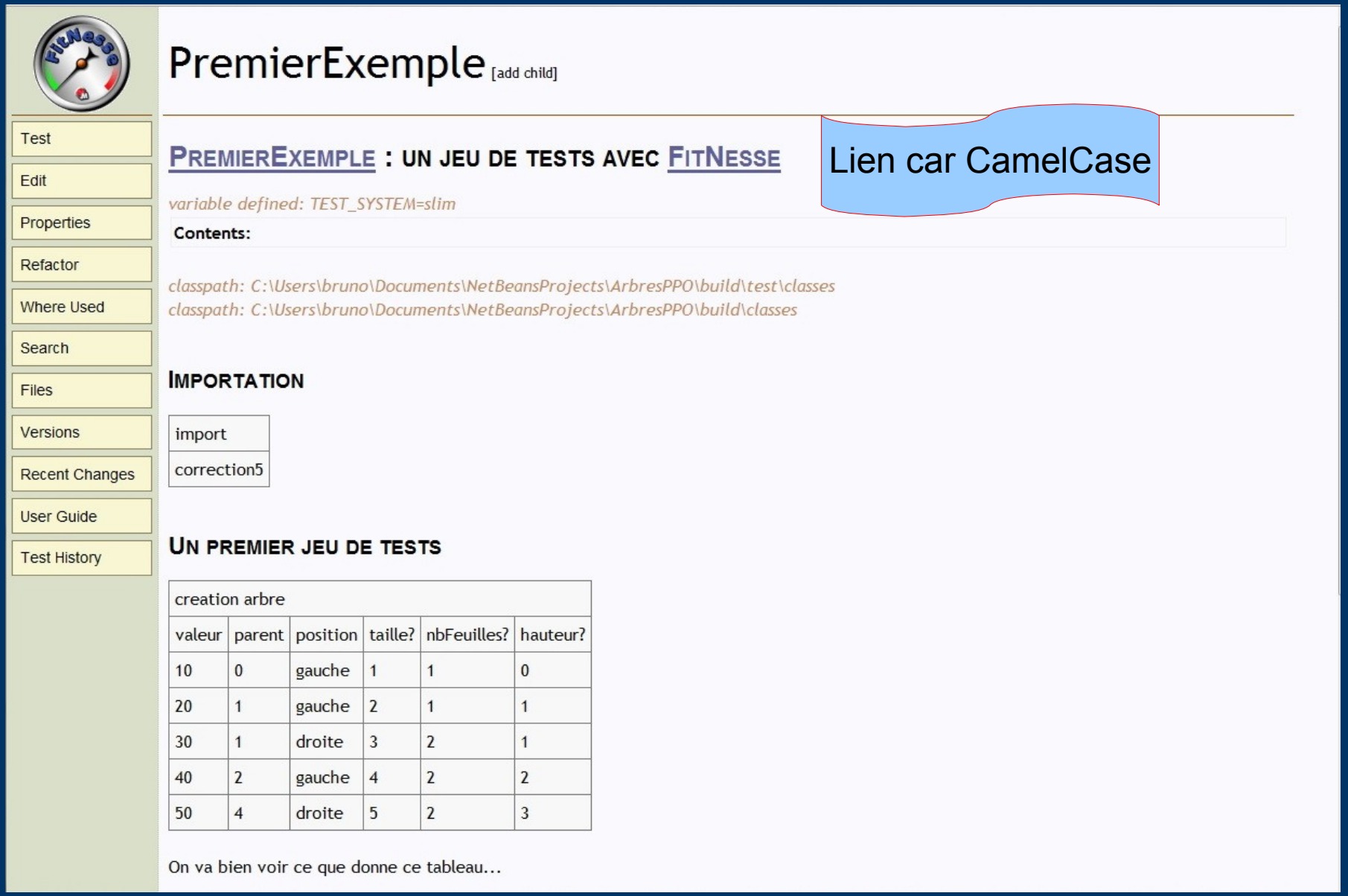
Table standard (« decision »)
Titre (creation arbre) = nom de la classe de « Fixture » (CreationArbre)
Colonnes sans « ? » : doivent correspondrent à des méthodes de la classe de Fixture : setValeur(int), setParent(int), setPosition(String)
Colonne avec « ? » : doivent correspondrent à des méthodes de la classe de fixture : int taille(), int nbFeuilles(), int hauteur().



Ne pas oublier de marquer dans les propriétés la page comme une page de test

(fait automatiquement si le nom de la page commence ou fini par Test ou Example)

Aperçu de la page précédente



Test
Edit
Properties
Refactor
Where Used
Search
Files
Versions
Recent Changes
User Guide
Test History

PremierExemple [add child]

PremierExemple : un jeu de tests avec FitNesse

Lien car CamelCase

variable defined: TEST_SYSTEM=slim

Contents:

classpath: C:\Users\bruno\Documents\NetBeansProjects\ArbresPPO\build\test\classes
classpath: C:\Users\bruno\Documents\NetBeansProjects\ArbresPPO\build\classes

Importation

import
correction5

Un premier jeu de tests

creation arbre					
valeur	parent	position	taille?	nbFeuilles?	hauteur?
10	0	gauche	1	1	0
20	1	gauche	2	1	1
30	1	droite	3	2	1
40	2	gauche	4	2	2
50	4	droite	5	2	3

On va bien voir ce que donne ce tableau...

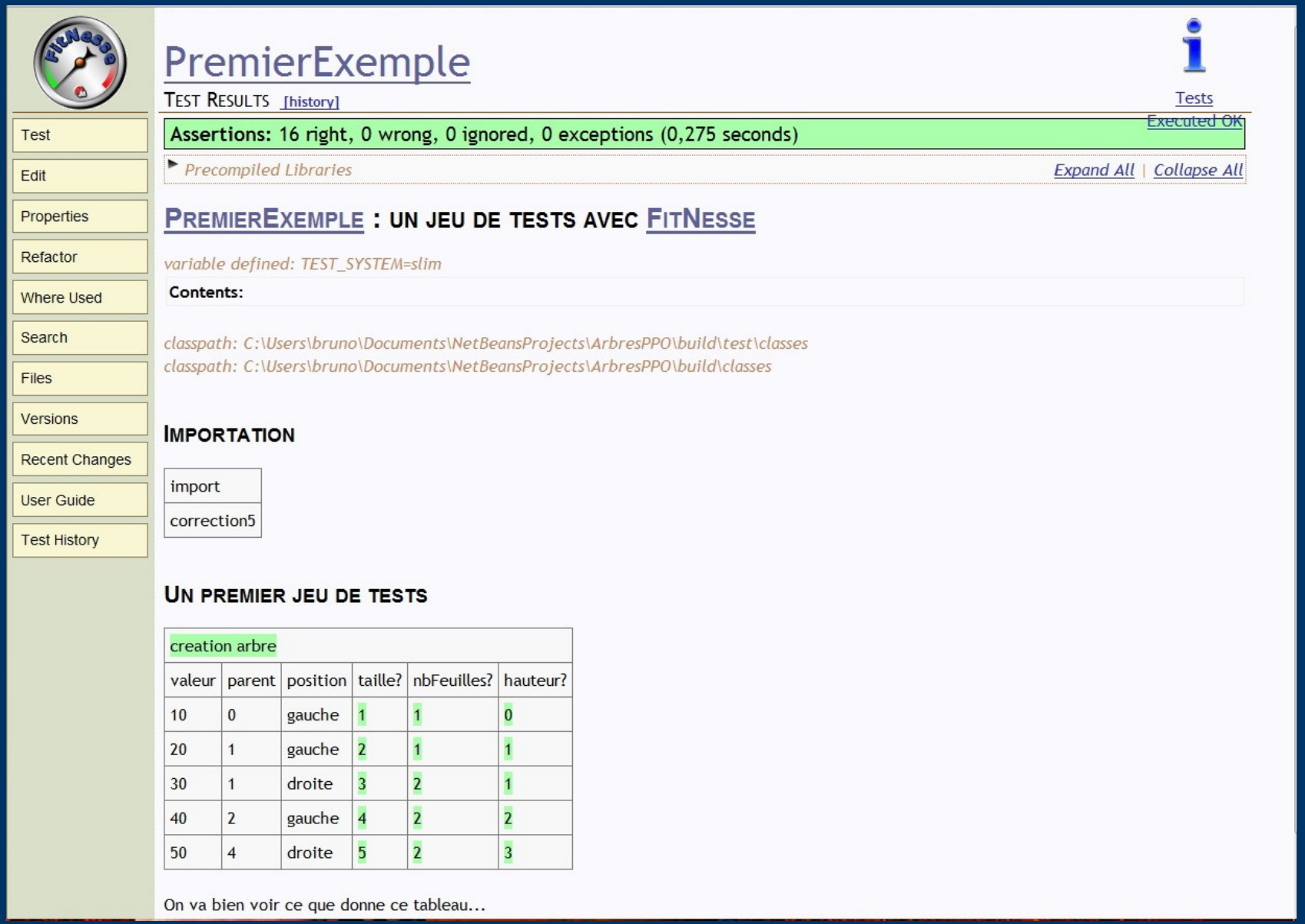
Classe de Fixture

```
package correction5;

import correction5.Arbre5.Position;

public class CreationArbre {
    private Arbre5<Integer> arbre;
    private int pere;
    private int valeur;
    private Position position;

    public CreationArbre() {}

    public void setValeur(int v) {valeur = v;}

    public void setParent(int p) {pere = p;}

    public void setPosition(String s) {
        if ("gauche".equals(s)) {
            position = Position.gauche;
        } else {
            position = Position.droit;
        }
    }
```

Méthode facultative appelée entre les « sets » et la collecte des résultats

```
    public void execute() {
        if (pere == 0) {
            arbre = new Arbre5<Integer>(valeur);
        } else {
            try {
                arbre.ajout(pere, valeur, position);
            } catch (NoeudInexistantException5 nie5) {
            }
        }
    }

    public int taille() {
        return arbre.taille();
    }

    public int nbFeuilles() {
        return arbre.nbFeuilles();
    }

    public int hauteur() {
        return arbre.hauteur();

    }
}
```

Résultat de l'exécution du test (après clic sur « Test »)



Test
Edit
Properties
Refactor
Where Used
Search
Files
Versions
Recent Changes
User Guide
Test History

PremierExemple

TEST RESULTS [history]

Tests Executed OK

Assertions: 16 right, 0 wrong, 0 ignored, 0 exceptions (0,275 seconds)

Precompiled Libraries

Expand All | Collapse All

PREMIEREXEMPLE : UN JEU DE TESTS AVEC FITNESSE

variable defined: TEST_SYSTEM=slim

Contents:

classpath: C:\Users\bruno\Documents\NetBeansProjects\ArbresPPO\build\test\classes
classpath: C:\Users\bruno\Documents\NetBeansProjects\ArbresPPO\build\classes

IMPORTATION

import
correction5

UN PREMIER JEU DE TESTS

creation arbre					
valeur	parent	position	taille?	nbFeuilles?	hauteur?
10	0	gauche	1	1	0
20	1	gauche	2	1	1
30	1	droite	3	2	1
40	2	gauche	4	2	2
50	4	droite	5	2	3

On va bien voir ce que donne ce tableau...

Quelques possibilités supplémentaires
Code à rajouter à la fin du précédent

!2 Un deuxième jeu de tests

|creation arbre2|
valeur	parent	position	taille?	nbFeuilles?	hauteur?	id?
10	0	gauche	1	1	0	$V10=
20	$V10	gauche	2	>=1	1	$V20=
30	$V10	droite	3	2	1	$V30=
40	$V20	gauche	4	2	3	$V40=
50	$V40	droite	5	2	3	

Nouvelle « fixture »

```
package correction5;

import correction5.Arbre5.Position;

public class CreationArbre2 {

    private Arbre5<Integer> arbre;
    private int pere;
    private int valeur;
    private Position position;

    public CreationArbre2() {}

    public void setValeur(int v) {valeur = v;}

    public void setParent(int p) {pere = p;}

    public void setPosition(String s) {
        if ("gauche".equals(s)) {
            position = Position.gauche;
        } else {
            position = Position.droit;
        }
    }

    public void execute() {
        if (pere == 0) {
            arbre = new Arbre5<Integer>(valeur);
        } else {
            try {
                arbre.ajout(pere, valeur, position);
            } catch (NoeudInexistantException5 nie5) {
            }
        }
    }

    public int taille() {return arbre.taille();}

    public int nbFeuilles() {return arbre.nbFeuilles();}

    public int hauteur() {return arbre.hauteur();}

    public int id() {
        try {
            return arbre.getIdDonnee(valeur);
        }
        catch (NoeudInexistantException5 nie5) {
            return 0;
        }
    }
}
```

FitNesse : Créer son premier test

- Mettre un nom (le nom de la page à créer) en CamelCase sur la page d'accueil en l'éditant, puis sauver
- Cliquer sur le « ? » à côté du nom en question
- Taper le contenu de sa page et sauver

Langage de marquage : référence

Mise en forme (1)

- "texte" (deux ') : italique
- "'texte"' (trois ') : gras
- --texte-- : barré
- !texte : centrage du texte
- !1, !2, !3 : titre de niveau 1, 2 ou 3
- !note texte : note
- ----, ------, -------- : ligne horizontale (au moins 4 tirets, épaisseur proportionnelle au nombre de tirets)
- !img URL, !img-l URL (texte), !img-r URL (texte) : image : insertion d'une image (ou toute url finissant en .gif ou .jpg)
- !-texte-! : évite l'interprétation wiki du texte
- !<texte>! : évite l'interprétation html du texte
- {{{texte}}} : texte pré-formatté

Mise en forme (2)

- Liste à puce : * texte, <esp>* texte, <esp><esp>*texte, etc.
- Liste numérotée : <num> texte, <esp>num texte, etc.
- Section condensable ouverte (au moins un '*')

```
!* titre
Texte
*!
```

- Section condensable fermée (au moins un '*')

```
!*> titre
Texte
*!
```

- Tables de texte (délimiteur=ponctuation quelconque)

```
![                    ![:
Ligne 1               champ11:champ12
Ligne 2               champ21:champ22
!]                    !]
```

Hyper-liens et date

- Liens internes
  - Marquage : !anchor nomDuLien
  - Référence : .#nomDuLien
- Lien externe : http://site/page
- Lien vers une autre page de FitNesse : CamelCase
- Alias de lien : [[alias][Lien]] où
  - Lien est l'un des types de lien possibles
  - Alias peut être du texte ou une image
- Date : !today, !today -t, !today – 1, !today (MMM), etc.

Types de table Slim : référence

Table « Décision »

- Syntaxe de base
  - |titre de la table| p1 | p2|
  - |col1 | col2 | col3 | col4? | col5?|
  - |val1 | val2 | val3 | val4 | val5|
  - |v1 | v2 | v3 | v4| v5|
- Interprétation
  - Va utiliser une « fixture » TitreDeLaTable
  - Un constructeur prenant 2 paramètres va être appelé avec p1 et p2
  - Si la méthode table(<List<List<String>>) est présente dans la fixture, elle est appelée avec une liste de listes contenant les valeurs dans la table
  - Pour chaque ligne de la table
    - Si la méthode reset() est présente, elle est appelée
    - Les méthodes setCol1(), setCol2() et set Col3() sont appelées avec les valeurs de la table
    - Si la méthode execute() est présente, elle est appelée
    - Les méthodes col4() et col5() sont appelées

Contenu des cases

- Valeur simple : test sur l'égalité
- != : différence
- < x, <= x, > x, >= x: inférieur à, supérieur à
- x <_< y : interval
- ~= x : valeur approchée (tolérance dépend de la précision de x)
- =~/regexp/ : Expression rationnelle

Table « requête »

- Syntaxe de base
  - |Query:titre de la table| p1 |
  - |col1 | col2 | col3 | col4 | col5|
  - |val1 | val2 | val3 | val4 | val5|
  - |v1 | v2 | v3 | v4| v5|
- Interprétation
  - Va utiliser une « fixture » TitreDeLaTable
  - Un constructeur prenant 1 paramètre va être appelé avec p1
  - Si la méthode table(<List<List<String>>) est présente dans la fixture, elle est appelée avec une liste de listes contenant les valeurs dans la table
  - La méthode List<Object> query() est appelée. Cette méthode retourne une liste de listes de listes : table de lignes de champs où un champ est une liste dont le premier élément est le nom du champ et le deuxième sa valeur
  - Le champ le plus à gauche sert d'identifiant de la ligne
  - On compare les lignes de la table avec ce qui est produit ; l'ordre des lignes n'est pas significatif
- Utilisation : pour vérification ou remplissage

Table « sous-requête »

- Syntaxe de base
  - |Subset Query:titre de la table| p1 |
  - |col1 | col2 | col3 | col4| col5|
  - |val1 | val2 | val3 | val4 | val5|
  - |v1 | v2 | v3 | v4| v5|
- Interprétation
  - Idem Query mais on demande juste que les lignes soient présentent dans le résultat

Table « requête ordonnée »

- Syntaxe de base
  - |Ordered Query:titre de la table| p1 |
  - |col1 | col2 | col3 | col4? | col5?|
  - |val1 | val2 | val3 | val4 | val5|
  - |v1 | v2 | v3 | v4| v5|
- Interprétation
  - Idem Query mais on demande en plus que les lignes soient dans le même ordre

Table « script »

- Exemple introductif

script	login dialog driver	Bob	xyzzy	
Login with username	Bob	and password	xyzzy	
check	login message	Bob logged in		
reject	login with username	Bob	and password	bad password
ensure	login with username and password;		Bob	xyzzy
note	Un commentaire			
show	number of login attempts			
$message=	login message			
start	login dialog driver	Kent	x123y	

- Appel d'une fonction
  - Syntaxe
    - Debut nom|param1|suiteNom|param2|suitNom2|param3|...
    - Nom;|param1|param2|param3
  - Lieu
    - Dans la première colonne
    - Après reject, ensure, show, $variable=

Table « script » : syntaxe détaillée (1)

- |script|nom de la classe|p1|p2|
  - script pour indiquer le type de table
  - p1, p2 : paramètres passés au constructeur
- Appel d'une fonction
  - Si fournit un résultat booléen, affichage vert/rouge après exécution
  - Sinon, laissé tel quel
- Check| appel d'une fonction | dernière case|
- Check not | appel d'une fonction | dernière case|

  Vérifie que le résultat fourni par l'appel de la fonction est identique/différent au contenu de la dernière case ; affichage vert/rouge

Table « script » : syntaxe détaillée (2)

- |ensure | appel d'une fonction |
- |reject | appel d'une fonction |

  La fonction doit fournir un résultat booléen, affichage vert/rouge après exécution

- |show | appel d'une fonction|

  Affichage de la valeur de retour de la fonction

- |$v=|appel d'une fonction|

  Affectation du retour de la fonction à la variable $v

- |start | nom de la classe | p1 | p2 | p3 |

  Création d'une nouvelle instance de la classe en appelant le bon constructeur et en lui passant les paramètres p1,p2 et p3. C'est cette instance qui sera utilisée pour la suite des tests.

Table « scenario »

- Principe
  - Table type « script » paramétrée
  - Appelable depuis une table « décision », une table « script » ou une autre table « scenario »
- Exemple (tiré de la doc. en ligne)

scenario	widget	wikiText	renders	htmlText
Create page	WidgetPage	with content	@wikiText	
check	request page	WidgetPage	200	
ensure	contentMatches	@htmlText		
show	content			

- Interprétation
  - Définition d'un scenario WidgetRender(wikiText, htmlText)
  - Les paramètres sont référencés via le caractère @ ; leur nom est en camel-case commençant par une minuscule

Table « scenario » : Exemple d'utilisation depuis une table Décision

- Préliminaires
  - Le scénario doit être défini dans la page de la table « décision »
  - Un scénario a priorité sur une « fixture » de même nom
- Exemple

widget renders		
wiki text	html text	
This is "italic" text	This is italic text	italic widget
This is '"bold"' text	This is bold text	bold widget

Utiliser FitNesse en résumé

- Les tests précèdent le code

  Ils formalisent les attentes du client, évitant aux développeurs de partir dans de fausses directions

- Mélanger texte informel et tables de tests

  Les tables ont pour but de formaliser le texte informel

- Faire collaborer client et fournisseur
  - Le client doit être à l'origine des pages et jeux de test
  - Le client doit pouvoir rajouter de nouveaux cas/jeux de tests à volonté
  - Le fournisseur doit pouvoir lancer les tests aussi souvent que possible (idem pour le client)