Salomon

System przetwarzania wiedzy

Test Report, ver 1.0

Historia wersji

Data	Wersja	Opis	Autorzy
19-09-2005	1.0	Raport testów	Tomasz Wąsala

1 Wstęp

1.1 Cel dokumentu

Dokument ma na celu przedstawienie raportu z testów implementacji drzew decyzyjnych an platformie Salomon. Dotyczy on wersji pre-release.

1.2 Referencje

Integralną częścią niniejszego opracowania są niżej wymienione dokumenty:

- Test Plan
- Software Architect Document

2 Narzędzia użyte do testów

Do tworzenia testów został użyty pakiet jUnit. Natomiast nie zdecydowaliśmy się na użycie Gringer'a.

Niektóre testy, np. testy działania pluginu obsługującego wizualizacje, będą też przeprowadzane ręcznie.

3 Opis testów

Zostały przetestowane następujące klasy należące do pakietu salomon.engine.platform.data.tree:

- Node
- DataSource
- Tree
- TreeManager

Pluginy zostały wstępnie przetestowane przez osoby tworzące je. Dokładniejsze testy będą zrobione wkrótce. Analogicznie podczas rozszerzania platformy zostały wykryte różne jej bugi, czego śladem są maile w naszej grupie dyskusyjnej.

Należy też tutaj wspomnieć, że testy w ramach poszczególnej klasy wzajemnie się uzupełniają i należy patrzeć na nie całościowo.

Testy zostały podzielone na dwie grupy: testy rozszerzenia platformy Salomon o obsługę drzew decyzyjnych; testy pluginów.

4 Przeprowadzone testy rozszerzenia platformy salomon

- **Klasa salomon.engine.platform.data.tree.Node**

  Są to testy na niskim poziomie abstrakcji sprawdzające poprawność zaimplementowania interfejsu salomon.platform.data.tree.INode. Specjalnie na pożytek poniższych testów zostały przygotowane węzły z przykładowymi danymi - testowe, do których będę się odwoływał przy opisach poszczególnych testów.

  - testAddChild

    Tworzone są Instancje klasy Node i dodawane do niej nowe węzły za pomocą metody addChild(). Następuje sprawdzenie wyników metodą getChildren().

    Wyniki:

    Pozytywne.

  - testAddChildren

    Tworzone są dwie tablice węzłów i dodawane są jako dzieci do jednego z węzłów. Poprawność dodawania sprawdzam poprzez porównanie typu, wartości i gałęzi rodzica z wartościami oczekiwanymi.

    Wyniki:

    Pozytywne.

  - testGetChildren

    Test polega na sprawdzaniu ilości pobranych dzieci z węzła testowego.

    Działanie tej metody ta jest też testowane w innych testach (np. testAddChild).

    Wyniki:

    Pozytywne.

  - testGetId

    Test polega na ustawieniu Id drzewa za pomocą metody setId(int), pobraniu ustawionego Id poprzez metodę getId(). Uzyskany Id jest porównywany z wartościami oczekiwanymi.

    Wyniki:

    Pozytywne.

  - testGetLeafs

    Test polega na pobraniu liści z węzłów testowych i porównaniu ilości otrzymanych liści z wartościami oczekiwanymi.

    Wyniki:

    Pozytywne.

- testGetParent

  Test polega na pobraniu rodziców z węzłów testowych i porównaniu otrzymanych wyników z wartościami spodziewanymi.

  Wyniki:

  Pozytywne.

- testGetParentEdge

  Test polega na pobraniu wartości krawędzi do rodzica od węzłów testowych za pomocą metody getParentEdge() i porównaniu otrzymanych wyników z wartościami oczekiwanymi.

  Wyniki:

  Pozytywne.

- testGetRoot

  Test polega na stworzeniu węzłów. W konstruktorach węzłów ustawiane są korzenie. Następnie pobieramy korzenie za pomocą metody getRoot() i porównujemy wyniki z wynikami spodziewanymi.

  Wyniki:

  Pozytywne.

- testGetType

  Test polega na sprawdzeniu wyników metody getType() z wartościami oczekiwanymi. Metoda getType() zostaje wywołana na węzłach testowych.

  Wyniki:

  Pozytywne.

- testGetValue

  Test polega na pobraniu "wartości" za pomocą metody getValue() z węzłów testowych i porównaniu z wartościami oczekiwanymi. Następuje tu też wywołanie metody setValue() i porównanie wyników getValue() po zmianach.

  Wyniki:

  Pozytywne.

- testIsLeaf

  Test polega na sprawdzeniu wyników funkcji isLeaf() zastosowanej na węzłach testowych.

  Wyniki:

  Pozytywne.

- testIsRoot

Test polega na sprawdzeniu wyników metody isRoot() wywołanje na węzłach testowych.

Wyniki:

Pozytywne.

- testNodeINodeStringTypeString

  Test konstruktora Node(INode, String, Type, String). Tworzymy kilka węzłów INode za pomocą konstruktora. I sprawdzamy wartości ich atrybutów z wartościami oczekiwanymi.

  Wyniki:

  Pozytywne.

- testNodeIntINodeStringTypeString

  Test konstruktora Node(int, INode, String, Type, String). Tworzymy kilka węzłów INode za pomocą konstruktora. I sprawdzamy wartości ich atrybutów z wartościami oczekiwanymi. Różni się od powyzszego sprawdzeniem wartości atrybutu id.

  Wyniki:

  Pozytywne.

- testSetChildren

  Test polega na dodaniu do węzła testowego jego dzieci. I porównaniu atrybutów otrzymanej struktury rodzic-dzieci z wartościami oczekiwanymi.

  Wyniki:

  Pozytywne.

- testSetId

  Test polega na stworzeniu tymczasowych węzłów, ustawieniu różnych wartości atrybutów Id za pomocą metody setId() oraz sprawdzeniu wyniów metody getId() z wartościami oczekiwanymi.

  Wyniki:

  Pozytywne.

- testSetParent

  Test polega na ustwaieniu węzłom testowym nowych rodziców i sprawdzeniu oczekiwanych wyników.

  Wyniki:

  Pozytywne.

- testSetParentEdge

  Test polega na ustawieniu wartości atrybutu parentEdge za pomocą metody setParentEdge(). Następnie zostają pobrane ustawione atrybuty i zostają sprawdzone z wartościami oczekiwanymi.

Wyniki:

Pozytywne.

– testSetType

Test polega na ustawieniu typu węzła testowego i następnie porównaniu wartości atrybutu Type z wartością oczekiwaną.

Wyniki:

Pozytywne.

– testSetValue

Test polega na ustawieniu wartości danego węzła i porównaniu wartości atrybutu value z wartościami oczekiwanymi.

Wyniki:

Pozytywne.

- **Klasa salomon.engine.platform.data.tree.TreeManager**

  – testGetTreeDataSourceData

  Poprzez odpowiednio spreparowane polecenia SQL zostały stworzone struktury w bazie danych umożliwiające pobranie DataSource'a. Polecenie tworzy w bazie danych rekord opisujący DataSourc'a korzystającego z fikcyjnej tabeli, która też została tymczasowo utworzona.

  Następnie zostaje tworzona tablica DataSource'ów, w której ostatnim elementem będzie właśnie DataSource stworzony na podstawie danych wpisanych przez nas do bazy danych. Następnie zostaje wywołana metoda getTreeDataSourceData(DataSource) i zawartość otrzymanej listy jest porównywana z wartościami spodziewanymi.

  Wyniki:

  Pozytywne.

  – testAddTree

  W tym teście tworzone są tymczasowe struktury w bazie danych opisujące datasource'a, zewnętrzną tabelę z danymi. Z przykładowych węzłów tworzymy proste drzewko. Z danych umieszczonych w bazie danych następnie tworzymy DataSource'a, którego przypisujemy do drzewa. Następnie dodajemy do bazy danych nasze drzewo za pomocą metody addTree(). Porównujemy strukturę i atrybuty tego drzewa i jego węzłów z wartościami oczekiwanymi. Porównujemy atrybuty także DataSource'a przypisanego do tego drzewa z wartościami oczekiwanymi.

  Wyniki:

  Problemy:

Podczas zapisywania drzewa do bazy danych w tabeli TREE_NODES dla węzła, który jest root'em wartość w kolumnie TRN_PARENT_NODE_ID jest ustawiana na 0, a powinno być NULL. Wynika to z braku obsługi NULL przez SQLInsert (salomon.engine.database.queries.SQLInsert). Ten błąd został też zgłoszony na naszej grupie dyskusyjnej.

5 Przeprowadzone testy pluginów i ich rezultaty

Testy pluginów zostały wstępnie przeprowadzone przez osoby je tworzące. Opis tych testów i dalsze testy zostaną umieszczone w tym punkcie.

5.1 Plugin - VeniTreeCreator - algorytmiczny

Pakiet - pl.edu.agh.capitol.veniTreeCreator.jUnit

Zostały przeprowadzone testy podstawowych klas logicznych należących do pluginu:

- pl.edu.agh.capitol.veniTreeCreator.logic.DataItem
- pl.edu.agh.capitol.veniTreeCreator.logic.TreeItem

oraz wysokopoziomowe testy środowiska dostarczonego przez platformę

- salomon.platform.IDataEngine
- salomon.platform.IEnvironment
- salomon.platform.IVariable

5.2 Plugin - - wejściowy

5.3 Plugin - TreeVis - wizualizacyjny