Testarea Unitara in BlueJ

Versiunea 1.0
Pentru BlueJ versiunea 1.3.0

Michael Kolling
Maersk Institute
University of Southern Denmark

Traducere de **Dumitrascu Irina** si **Popa Adrian**

Copyright © M. Kolling

Copyright © M. Kolling

1 Introducere

Rezumat: BlueJ ofera posibilitatea testarii regresive a functionalitatii integrand JUnit.

1.1 Despre acest tutorial – audienta tinta

Acest tutorial introduce functionalitatea testarii unitare in mediul BlueJ. Presupunem ca dvs cunoasteti deja functionalitatea de baza a mediului BlueJ. Daca nu, cititi inainte 'Tutorialul BlueJ'. (Puteti obtine acel tutorial, si o versiune electronica a acestuia la adresa http://www.bluej.org/doc/documentation.html).

De asemenea, presupunem ca sunteti familiar cu notiunea de testare unitara (unit testing) (sau cel putin cu testarea software-ului in general). Va dam mai multe indicii in sectiunea urmatoare.

1.2 Ce este testarea unitara?

Termenul de 'testare unitara' se refera la testarea individuala a unor unitati separate dintr-un sistem software. In sistemele orientate spre obiecte, aceste 'unitati' sunt de regula clase si metode. Astfel, in contextul nostru, testarea unitara se refera la testarea individuala a metodelor si claselor in BlueJ.

Acest tutorial se ocupa de uneltele BlueJ care permit testarea unitara *sistematica*. Daca sunteti familiar cu posibilitatiile de interactiune cu BlueJ, stiti deja ca este usor sa testezi metode individuale interactiv. Vom numi acest mod *testare ad-hoc*.

Testarea ad-hoc este buna, dar insuficienta pentru a face testari sistematice. Elementele de testare unitara din BlueJ va pun la dispozitie unelte pentru a inregistra si repeta teste, pentru ca testele unitare sa poata fi repetate usor mai tarziu (de regula cand se schimba o parte din sistem), astfel incat dezvoltatorul sa fie convins ca noile modificari nu au stricat vechea functionalitate. Acest lucru e cunoscut ca *testare regresiva*.

Conceptele testarii unitare si testarii regresive sunt destul de vechi, dar popularitatea lor a crescut brusc de curand, dupa publicarea *metodologiei de programare eXtreme*[1] si dupa aparitia unei unelte de testare unitara pentru Java: *JUnit*.

JUnit este un cadru de testare regresiva scris de Erich Gamma si Kent Beck. Puteti gasi software-ul si o multime de informatii despre el la adresa: *http://www.junit.org*.

1 Pentru a afla ce inseamna programare eXtrema, uitati-va prin *"Extreme Programming Explained: Embrace Change", Kent Beck, Addison Wesley, 1999.* Mai sunt multe alte carti bune disponibile. Un ghid bun online este disponibil la: http://www.xprogramming.com/xpmag/whatisxp.htm

1.3 Testarea unitara in BlueJ

Uneltele pentru testarea sistematica din BlueJ sunt bazate pe JUnit. Astfel, cunostiintele generale despre JUnit va vor ajuta sa intelegeti testarea unitara din BlueJ. Va recomandam sa cititi un articol despre acest lucru (nu chiar acum, dar mai tarziu). Exista multe astfel de articole iar site-ul JUnit e un punct bun de plecare pentru a le gasi.

Testarea unitara din BlueJ combina capacitatea BlueJ de a face testare interactiva cu suportul pentru testare regresiva oferit de JUnit. Ambele moduri de testare sunt suportate complet. In plus, este suportata si o functionalitate noua pe baza combinarii celor doua moduri de testare: de exemplu, pot fi inregistrate secvente de test interactive pentru a creea automat metode de test JUnit ce pot fi folosite in teste regresive. Exemple in acest sens sunt date mai tarziu in acest document.

Functionalitatea testarii unitare in BlueJ a fost proiectata si adaugata de Andrew Patterson (Universitatea Monash) ca parte a lucrarii sale de doctorat.

Copyright © M. Kolling

2 Activarea functionalitatii pentru testare unitara

Rezumat: Uneltele de testare pot fi facute vizibile prin activarea unui parametru in preferinte

Suportul explicit pentru testare unitara in BlueJ este dezactivat initial. Pentru a folosi uneltele de testare folositi *Tools – Preferences...* si selectati casuta intitulata *Show testing tools*.

Activarea acestei functionalitati duce la aparitia a trei elemente noi in interfata: cateva butoane si un indicator de inregistrare in bara cu unelte a ferestrei principale, un element *Show test results* in meniul *View* si o inregistrare noua *Create Test Class* in meniul context a unei clase compilate.

Copyright © M. Kolling

3 Crearea claselor de test

Rezumat: Creati o clasa de test selectand Create Test Class *din meniul context al clasei.*

Primul pas in pregatirea unei clase sau metode pentru testare este crearea unei clase de test.

O clasa de test este o clasa asociata cu o clasa de proiect (pe care noi o vom numi *clasa de referinta*). Clasa de test contine teste pentru metodele clasei referinta.

Pentru exemplele din acest tutorial vom folosi proiectul *people*, distribuit impreuna cu BlueJ ca un exemplu din directorul *examples*. Poate ca doriti sa-l deschideti pentru a incerca diverse lucruri prezentate in continuare.

Puteti creea o clasa de test dand click dreapta pe o clasa compilata (pentru MacOS trebuie sa dati CTRL+Click) si sa selectati *Create Test Class* din meniul context. Clasa de test este numita automat adaugand *Test* la numele clasei de referinta. De exemplu, daca numele clasei de referinta este *Student,* atunci clasa de test va fi numita *StudentTest*.

Clasele de test sunt inidcate in diagrama impreuna cu tag-ul <<*unit test*>>. Ele au si o culoare diferita (Fig 1). Repozitionarea clasei de referinta va face ca clasa test sa ramana atasata.

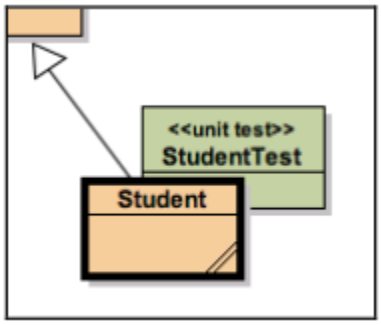

Fig 1: O clasa referinta cu o clasa de test atasata.

Clasele de test sunt tratate intr-un mod special de mediu. Ele au functiunile specifice claselor (cum ar fi *Open Editor, Compile, Remove*), dar si functiuni specifice testelor (Fig 2). Clasa de test trebuie sa fie compilata pentru a fi vizibil acest meniu.

Copyright © M. Kolling

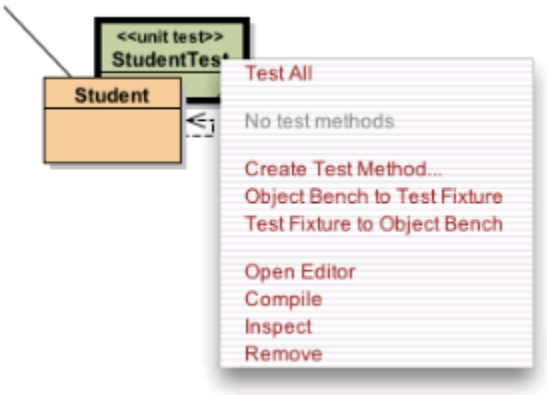

Fig 2: Meniul context al unei clase de test.

Crearea unei clase de test nu duce si la crearea testelor, dar acum avem posibilitatea de a creea teste. Clasa de test este doar un container pentru testele pe care le vom creea.

Copyright © M. Kolling

4 Crearea metodelor de test

Rezumat: Creati o metoda de test selectand Create Test Method... *din meniul clasei de test.*

Obiectele de tip *Student* au doua metode, *setName* si *getName* (mostenite de la *Person*), pentru a stabili si a afla numele studentului. Sa presupunem ca dorim ca creem un test pentru a vedea daca aceste metode merg cum trebuie.

Incepem prin a selecta *Create Test Method...* din meniul clasei *StudentTest*. O metoda de test implementeaza un singur test (adica testeaza doar o singura functionalitate).

Dupa ce ati facut selectia, veti fi intrebat de numele testului. Numele testelor incep intotdeauna cu prefixul *test* – daca numele ales de dvs nu incepe cu *test*, aceasta particula va fi adaugata automat. Astfel, tastarea *testName* sau *name* va duce la crearea unei metode de test numita *testName*.

Dupa ce ati tastat numele si apoi ati dat Ok, toate interactiunile vor fi inregistrate ca parte a acestui test. Indicatorul de inregistrare este pornit si sunt activate butoanele de oprire sau anulare a testului (Fig 3).

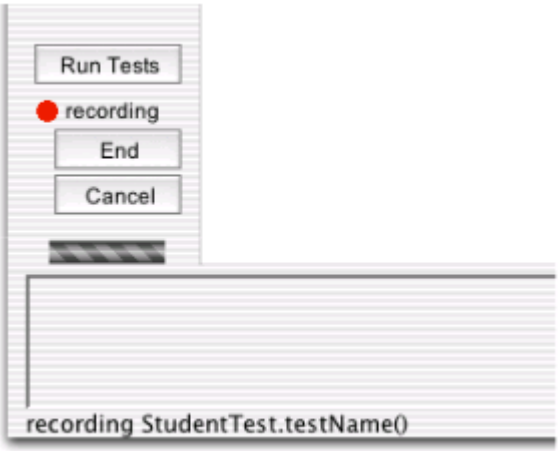

Fig 3: Starea butoanelor in timpul inregistrarii

Pentru a inregistra un test in acest exemplu, faceti urmatorii pasi:

- creati un obiect de tip *Student* folosind constructorul fara parametrii
- apelati metoda *setName(newName)* (mostenita de la *Person*) si setati numele la “Fred”
- apelati metoda *getName()*

Dupa apelul metodei *getName* veti vedea fereastra cu rezultate. Cata vreme inregistram teste, fereastra cu rezultate include o parte care ne permite sa atribuim conditii rezultatelor (Fig 4). Putem folosi aceste conditii pentru a specifica raspunsul asteptat ca rezultat al testului. In cazul nostru ne asteptam ca rezultatul testului sa fie egal cu string-ul “Fred”, asa ca putem impune aceasta conditie (Fig 4).

Copyright © M. Kolling

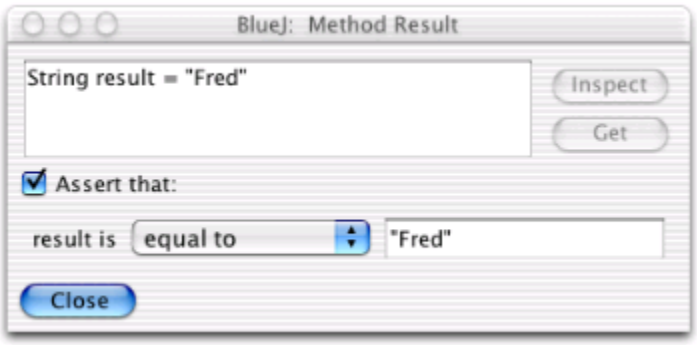

Fig 4: Fereastra de rezultate, cu optiuni asupra conditiilor

Exista diferite conditii ce pot fi impuse rezultatelor disponibile prin meniul context, inclusiv teste de egalitate, *null* si *non-null*.

Noi am terminat testul nostru, astfel incat putem apasa *End* pentru a incheia inregistrarea testului.

Incheierea testului duce la adaugarea unei metode de test in clasa de test. Aceasta metoda de test poate fi rulata.

Putem folosi butonul *Cancel* pentru a anula o inregistrare si a o elimina.

Intr-un mod similar cu acest exemplu, putem inregistra o multime de teste. Fiecare clasa din proiect isi poate avea clasa de test si fiecare clasa de test poate implementa oricate teste.

Fiecare inregistrare a unui test poate sa contina un numar oarecare de actiuni, inclusiv creare de obiecte sau oricate conditii asupra valorilor returnate.

Copyright © M. Kolling

5 Rularea testelor

Rezumat: Rulati toate testele apasand butonul Run Tests. *Rulati testele individuale selectandu-le din meniul context al clasei*.

Odata ce au fost inregistrate, testele pot fi rulate. Clasele de test sunt si clase Java, ca si clasele de referinta, astfel incat si ele trebuiesc compilate inaintea executiei.

BlueJ incearca automat sa compileze clasele de test dupa inregistrarea fiecarui test. Daca expresia de verificare contine erori sau daca clasa de test a fost editata de mana, va fi nevoie sa compilati explicit clasa de test inainte de a o putea folosi.

Acum putem da click dreapta pe clasa de test si vom vedea testul pe care l-am inregistrat in meniul context. Fig 5 prezinta un exemplu cu metoda *testName* si o a doua metoda de test numita *testStudentID*.

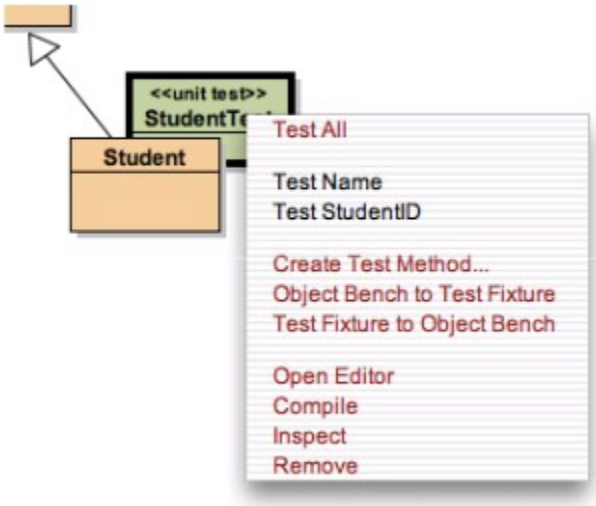

Fig 5: Meniul clasei de test cu doua metode de test definite.

Selectarea unui test din meniu duce la executia individuala a acelui test. Selectarea optiunii *Test All* din meniu duce la rularea tuturor testelor definite in acea clasa.

Cand un test este rulat individual, se vor intampla unul sau doua lucruri: daca testul se incheie cu succes (verificarile sunt adevarate), un mesaj scurt va va informa despre succes in bara de status a ferestrei proiectului. Daca testul se incheie cu esec (ori nu se respecta o verificare, ori apare alta problema), va fi afisata o fereastra de test prezentand detalii despre acest test (Fig 6).

Daca se ruleaza toate testele, atunci va aparea tot timpul o fereastra pentru a afisa starea testelor.

Puteti folosi si butonul *Run Tests* de deasupra indicatorului de inregistrare din fereastra principala. Activarea acestui buton duce la rularea tuturor testelor din toate clasele de test din pachet. Acesta e modul standard in care se executa un test complet pentru pachet.

Copyright © M. Kolling

6 Interpretarea rezultatelor testelor

Rezumat: Fereastra "Test results" afiseaza un rezumat al testelor efectuate si poate da detalii despre esecuri.

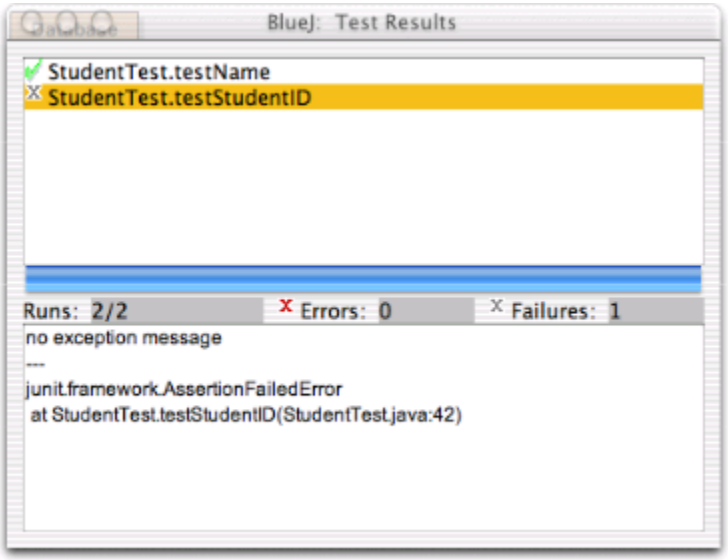

Fig 6: Fereastra Test Results

Dupa ce au fost rulate testele, fereastra *Test Results* afiseaza rezultatele (Fig 6). Partea superioara a ferestrei afiseaza o lista cu toate testele executate, impreuna cu o pictograma care indica succesul sau insuccesul lor. Un semn 'vazut' verde indica faptul ca testul s-a terminat cu succes, o cruciulita gri indica faptul ca testul s-a terminat cu esec, iar o cruciulita rosie indica o eroare.

Numarul de teste rulate, erori si esecuri este indicat si in centrul ferestrei.

Un test va avea o eroare daca in timpul executiei sale se genereaza orice tip de eroare, de genul unei exceptii aruncate si netratate.

Pentru toate testele care se termina cu esec, se pot viziona detalii despre cauza eseculi, selectand testul din lista. Jumatatea de jos a ferestrei va afisa informatii detaliate despre cauza erorii.

Bara din mijlocul ferestrei de test este rezumatul principal al rularii testelor: daca este verde, atunci totul este in ordine – toate testele s-au incheiat cu succes. Daca apare rosu, este o problema – cel putin un test a esuat.

Atentie: Pe MacOS aceasta bara nu-si schimba culoarea.

Copyright © M. Kolling

7 Ce este o fixtura?

Rezumat: O fixtura de test este un set de obiecte pregatit ca un punct de start pentru teste.

Cateodata unele teste au nevoie ca anumite obiecte sa fie in anumite stari inainte sa inceapa testul. De exemplu, cateva teste vor implica clasa *Database* din proiectul *people*, astfel incat avem nevoie de un obiect *Database*, un obiect *Student* si un obiect *Staff*. In plus dorim sa specificam starea pentru obiectul *Student* si cel *Staff* (adica sa setam un nume si o varsta).

Am putea sa incepem fiecare test prin crearea obiectelor individuale si trecerea lor in starile corespunzatoare, dar pe masura ce testele devin mai sofisticate, acest lucru poate deveni dificil si putem folosi un alt mecanism pentru a scapa de aceasta complexitate.

Putem creea o stare in object bench (un set de obiecte, fiecare intr-o stare) pe care dorim sa o folosim ca punct de plecare pentru toate testele dintr-o clasa de teste. Setul initial de obiecte e numit *fixtura*

Fixturile pot fi definite si ele vor fi aduse automat in starea necesara la inceputul fiecarui test din acea clasa de test, reducand astfel munca depusa pentru fiecare test.

Copyright © M. Kolling

8 Crearea si folosirea fixturilor de test

Rezumat: Pentru a creea o fixtura pentru o clasa de test, creati obiectele necesare in object bench si selectati Object Bench to Test Fixture *din meniul clasei de test.*

Incepem sa creem o fixtura de test prin crearea obiectelor de care avem nevoie, si apeland metodele obiectelor pentru a le aduce in starea necesara.

De exemplu, pentru a testa baza de date din proiectul *people*, dorim un obiect *Database*, un obiect *Student* cu numele *Fred,* pe care il vom insera in baza de date, si un obiect *Staff* cu numele *Jane* care nu va fi in baza de date. Putem creea obiectele si sa facem apelul necesar pentru a-l insera pe Fred in baza de date.

De indata ce obiectele din object bench sunt in starea pe care o dorim inainte de a incepe testul putem selecta functia *Object Bench to Test Fixture* din clasa *DatabaseTest*.

Selectarea acestei functii va creea o fixtura de test pentru clasa si va inlatura obiectele din object bench.

Cand o clasa are asociata o fixtura, aceasta fixtura va fi recreeata la inceputul fiecarui test. De exemplu, daca vrem sa creeam un test nou pentru clasa *Database* (selectand *Create Test Method* din clasa de test), starea definita in fixtura va fi automat recreata. Obiectele din fixtura vor aparea in starea lor definita in object bench inainte de inceperea inregistrarii testului.

Starea fixturii poate fi analizata in object bench selectand *Test Fixture to Object Bench* din meniul clasei de test. Acest lucru e util in cazurile in care dorim sa extindem fixtura mai tarziu, deoarece metodele noi de test au nevoie de obiecte noi.

In acest caz, vom folosi *Test Fixture to Object Bench* pentru a regenera starea fixturii, si apoi vom face modificari manual pentru a aduce fixtura in noua stare dorita. Dupa ce am terminat, putem selecta *Object Bench to Test Fixture* pentru a salva fixtura. Vechea fixtura va fi inlocuita.

Copyright © M. Kolling

9 Scrierea metodelor de test de mana

Rezumat: Metodele de test pot fi scrise direct in codul sursa al clasei de test.

Generarea metodelor de test si a fixturilor prin inregistrarea interactiunii cu clasele si obiectele este doar o optiune in generarea testelor unitare. Cealalta optiune este sa scrieti aceste metode de test de mana.

Clasele de test sunt clase Java, ca si celelalte clase din proiect, si in consecinta pot fi tratate ca orice alte clase. In mod special, putem deschide editorul sa modificam codul sursa, putem compila si rula codul sursa.

In modul traditional (non-BlueJ) de a folosi JUnit, acesta este modul standard de a creea metode de test, iar BlueJ permite si acest mod de lucru. Inregistrarea interactiva a testelor este o alternativa la scrierea lor de mana, nu o inlocuire.

Pentru persoanele care nu sunt obisnuite cu JUnit, un punct bun de plecare este generarea unei fixturi de test si crearea in mod interactiv a catorva metode de test si apoi sa se examineze codul sursa al claselor de test. Se va observa ca fiecare clasa de test are o metoda numita *SetUp()* care este folosita pentru a realiza o fixtura de test. In plus mai are cate o metoda pentru fiecare test.

Este permisa modificarea de mana a testelor existente, sau adaugarea unor metode de test complet noi, scrise de mana. Tineti minte ca numele unei metode de test trebuie sa inceapa cu prefixul *test* pentru a fi recunoscuta de mediu ca o metoda de test.

Cei care doresc sa stie mai multe despre cum se scriu teste JUnit ar trebui sa citeasca materialele recomandate la sfarsitul acestui tutorial.

Copyright © M. Kolling

10 Scrierea testelor

Rezumat: Pentru a creea teste inaintea implementarii, testele pot fi scrise de mana sau se pot folosi cioturi de metode.

Metodologia Programarii eXtreme sugereaza ca testele trebuiesc scrise *inainte* de a scrie implementarea oricarei metode. Folosind tool-urile pentru testare unitara oferite de BlueJ, acest lucru poate fi facut in doua moduri.

In primul rand, testele pot fi scrise de mana, asa cum a fost prezentat in sectiunea anterioara. Scrierea testelor se va face la fel ca intr-un mediu JUnit diferit de BlueJ.

In al doilea rand putem scrie *cioturi* de metode in clasa referinta, care sa returneze diverse valori pentru metodele de tip diferit de *void*. Dupa asta putem creea teste folosind facilitatea de inregistrare interactiva si sa scriem conditiile de test conform cu asteptarile de la versiunea cu implementare.

Copyright © M. Kolling

11 Testarea multi-clasa

Rezumat: Optiunea New class... *cu tipul clasei setat la* Unit Test *poate fi folosita pentru a creea clase de test neatasate.*

Exemplele afisate mai sus folosesc clase de test atasate unor clase de referinta. Aceasta atasare nu impiedica clasa de test sa foloseasca obiecte de tipul celorlalte clase din proiect in teste, dar sugereaza o conexiune logica a clasei de test cu clasa referinta.

Uneori se scriu clase de test care testeaza interactiunea mai multor clase. Aceste clase nu apartin in mod logic de o singura clasa. Astfel, ele nu ar trebui sa fie atasate in mod direct de o singura clasa.

Putem creea clase neatasate selectand *New Class...* si apoi selectand *Unit Test* ca tip al clasei.

Clasele de test neatasate pot fi folosite in acelasi mod ca si celelalte clase de test. Putem creea fixturi de test, putem face metode de test si sa rulam teste.

Copyright © M. Kolling

12 Rezumatele

1. *BlueJ ofera posibilitatea testarii regresive a functionalitatii integrand JUnit.*
2. *Uneltele de testare pot fi facute vizibile prin activarea unui parametru in preferinte*
3. *Creati o clasa de test selectand* Create Test Class *din meniul context al clasei.*
4. *Creati o metoda de test selectand* Create Test Method... *din meniul clasei de test.*
5. *Rulati toate testele apasand butonul* Run Tests. *Rulati testele individuale selectandu-le din meniul context al clasei*.
6. *Fereastra "Test results" afiseaza un rezumat al testelor efectuate si poate da detalii despre esecuri.*
7. *O fixtura de test este un set de obiecte pregatit ca un punct de start pentru teste.*
8. *Pentru a creea o fixtura pentru o clasa de test, creati obiectele necesare in object bench si selectati* Object Bench to Test Fixture *din meniul clasei de test.*
9. *Metodele de test pot fi scrise direct in codul sursa al clasei de test.*
10. *Pentru a creea teste inaintea implementarii, testele pot fi scrise de mana sau se pot folosi cioturi de metode.*
11. *Optiunea* New class... *cu tipul clasei setat la* Unit Test *poate fi folosita pentru a creea clase de test neatasate.*

Copyright © M. Kolling