Tema laborator 5
Interfete I. TDD.

In aceasta tema veti face primii pasi intr-o tehnica de programare deosebit de utila, „test driven development”. Veti duce practica testelor la un nivel un pic mai elevat.

Va veti juca cu interfetele si veti vedea ce usor se folosesc.

In sfarsit vom rezolva elegant toate problemele ridicate in tema 3 (functii non polinomiale si polinomiale, la care le gasim radacina cu aceeasi functie Newton)

1 Materiale aditionale ce trebuie studiate

Cititi si intelegeti (de pe : http://users.utcluj.ro/~jim/OOPR/lab-announce.html)

Laborator 6: Interfete. (Fara 2.4)

2 Teorie

Conceptul de incapsulare este concretizat prin clase. Intr-o clasa se incapsuleaza datele impreuna cu codul ce manipuleaza aceste date. Obiectele sunt instantieri ale clasei, fiecare obiect avand zona lui de memorie privata.

Clasele sunt ca niste sabloane dupa care se creaza obiectele.

Interfetele sunt niste contracte intre doua parti. Clasa/obiectul care implementeaza o interfata SE OBLIGA sa respecte toate conditiile impuse de interfata, iar cel ce foloseste interfata poate folosi oricare din metodele puse la dispozitie de interfata.

Clasele pot implementa oricate interfete. Interfetele in sine NU contin nici date si nici cod.

3 Practica

Continuati pe tema numarul 4. Ar trebui sa aveti implementata o clasa Func ce accepta un polinom de ordin general.

3.1 Crearea unei interfete

Creati o interfata numita Function ce sa contina metodele getFuncValue si getDerivValue cu aceeasi semnificatie ca si in laboratorul 3.

(Shortcut: Puteti “extrage” o interfata dintr-o clasa existenta. Click dreapta pe clasa, (ex pe Func) si selectati Refactor -> extract interface. Aici, dati numele interfetei si selectati ce metode sa aiba noua interfata. In cazul nostru ne intereseaza getFuncValue si getDerivValue)

Rulati testele sa vedeti daca ati gresit la ceva.

3.2 Utilizarea interfetei

Faceti “refactoring” la metoda RootFinding.findRoots() astfel incat sa opereze pe interfata Function

Rulati testele acestei clase si asigurati-va ca findRoots() functioneaza corespunzator.

(Obs. Daca ati folosit Refactor-> extract interface este posibil ca Eclipse sa fi facut "refactoring" deja la metoda findRoots)

3.3 Modifcarea implementarii Func

Modificati clasa Func astfel incat sa implementeze interfata Function.

Asigurati-va ca TOATE testele merg!

Deasemenea rulati si codul din metoda main().

In acest moment, ati rescris proiectul de la tema 3 folosind interfetele. Dupa cum ati observat, nu a trebuit sa faceti modificari majore in cod. Sa vedem ce avantaje aduce lucrul cu interfete.

3.4 Test Driven Development

Vom crea o noua clasa (numita LogFunc), care poate deriva functii logaritmice. Clasa implementeaza interfata Function. Matematic, va implementa f(x) = a*ln(x) Ca membri, avem constanta *a*. Pentru a calcula logaritmul unei valori y, scrieti Math.log(y). ln(x) derivat = 1/x.

Inainte de a face clasa, vom scrie teste pentru aceasta clasa. Aceasta practica este destul de raspandita in industrie (mai multe, la referinte)

Din cauza ca NU avem o clasa pentru care sa scriem teste, mergem in folderul de test, pachetul ro.utcluj.poo.lab05 si cream clasa LogFuncTest.

1. Adaugam metoda de test simpla, ce instantiaza un obiect LogFunc cu valoarea 1 pentru parametrul *a* si calculeaza valoarea functiei pentru un punct (pentru x=1 trebuie sa dea 0, oricare ar fi valoarea lui a) Observam ca nu compileaza nimic. (Obs: Nu uitati sa o adnotati cu @Test)
2. Acum trecem la scrierea clasei LogFunc in folderul src, pachetul ro.utcluj.poo.lab05. Scriem declaratia, variabilele membru, constructorul si metodele. La fiecare metoda scriem return 0.
3. Rulam testele, deci pana aici suntem OK

Mai facem o iteratie:

4. Adaugam un test pentru un alt punct. Observam ca testul esueaza.
5. Scriem cod in getFuncValue, testul trebuie sa treaca

Mai facem 1-2 iteratii pentru getDerivValue

In acest moment avem cateva functii de test cu cod identic, dar cu date diferite. Pachetul TestNG ne pune la dispozitie mecanismul de DataProvider. Ideea este ca avem o metoda care genereaza date de intrare pentru metoda care efectiv testeaza.

1. Cream metoda DataProvider. Aceasta are un format specific:

Trebuie sa returneze un array bidimensional de Object;

Trebuie sa nu aiba parametri de intrare

Trebuie sa fie adnotata cu @DataProvider

Poate sa aiba orice nume (dar sa fie publica)

In tabelul bidimensional, avem cate un rand pentru fiecare caz de test. Fiecare rand contine cateva coloane, coloane care vor constitui parametri de intrare la metoda de test

Exemplu:

```
@DataProvider
public Object[][] valoriDeTest(){
  return new Object[][]{
          {1.0,0.0},
          {4.0, 10.0},
  };
}
```

2. Cream metoda care efectiv testeaza codul

Aceasta metoda este o metoda de test normala, doar ca dupa adnotare scriem (dataProvider = "nume metoda ce genereaza date") si faptul ca metoda accepta parametri la intrare. Atentie, numarul parametrilor de intrare trebuie sa fie identic cu numarul de coloane din tabelul generat!

```
@Test(dataProvider = "valoriDeTest")
public void testMethd_1(double input, double expected) {
  System.out.println(input+" "+expected);
}
```

Putem avea oricati parametri la intrare, si oricate suite de test.

Acum putem rescrie codul de test astfel incat sa avem doar 2+2 metode, pentru toate punctele testate (2 metode de test efectiv, una pentru getFuncValue si una pentru getDerivValue si 2 metode ce genereaza datele).

O alta imbunatatire este sa cream o metoda adnotata cu @BeforeTest/@BeforeClass ce va crea o singura instanta a obiectului LogFunc.

3.5 Functie sinusoidala

Implementati tot in stilul TDD, f(x) = a*sin(b*x), numiti clasa SinFunc. De data aceasta puteti “scurtcircuita” putin tehnica TDD si sa incepeti direct cu @DataProvider

3.6 Operarea cu obiecte

Acum trebuie sa verificati daca metoda originala de gasire a radacinii functioneaza cu noile clase.

La nivel de cod, metoda findRoots() nu ar mai trebui atinsa.

La fel ca si in laboratorul anterior, creati un tabel de tip Function pe care il populati cu diverse tipuri de functii: Func, LogFunc, SinFunc.

Apelati metoda ce gaseste radacinile pentru un array de functii si observati rezultatele. Ar trebui sa va faceti pe hartie/calculator cateva exemple de test inainte.

In mod normal, testele de la cap 3.6 trebuie sa fie de sine statatoare, in niste clase de test. Deocamdata le rulam in main().

4 Evaluare

Tema se considera incheiata cand ati executat toti pasii din laborator si toate testele sunt verzi

5 Referinte:

http://en.wikipedia.org/wiki/Test-driven_development