Tema laborator 4
Manipularea obiectelor. Tabele. Bazele testarii

In aceasta tema extindeti clasa Func astfel incat sa memoreze polinoame de ordin general. Invatati deasemenea bazele testarii si lucrului cu array-uri.

1 Materiale aditionale ce trebuie studiate

Cititi si intelegeti (de pe : http://users.utcluj.ro/~jim/OOPR/lab-announce.html)
Laborator 5: Clase, obiecte şi tablouri (tot)
Introducerea de la cateva tutoriale de testare (vezi referintele)

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.

Functia polinomiala de gradul n are forma $f(x) = a_0x^n+a_1x^{n-1}+ ...+a_n$.
Derivata are forma $f'(x) = n*a_{0*}x^{n-1}+(n-1)a_1x^{n-2}+ ... a_{n-1}$

2.1 Arrays in java:

Un array se declara folosind notatia []. Exemplu, pentru un array de integers, lungime 5:

```
int[] tabel = new int[5];
```

Accesul se face de la indexul 0. Ex:

```
tabel[3] = 4;
k = tabel[i] * 2;
```

Lungimea poate fi aflata folosind tabel.length, Ex:

```
l = tabel.length;
```

2.2 Teste:

Detalii mai multe despre testarea unitara se vor face intr-un curs viitor. Deocamdata este necesar sa stiti ca testele regresive se pot executa de ori cate ori vrem si sunt vitale pentru a ne asigura ca programul inca functioneaza in urma modificarilor aduse.

Utilitarul pentru testare se numeste TestNG. Clasa ce contine testele este o clasa java normala cu cateva adnotari. (Textul cu @, situat inaintea declararii metodei). Utilitarul “vede” aceste metode si le executa. El mai observa daca testul esueaza sau nu si raporteaza acest lucru.

Ce sunt adnotarile, aici: (http://docs.oracle.com/javase/tutorial/java/annotations/). Esenta este ca sunt un fel de “comentarii” pe steroizi. Ele incep cu @ urmate de un fel de cuvant cheie.

In continuare, crash course in cum se face practic testarea:

Instrumentul de baza al testarii este o metoda publica, de tip void, fara nici un parametru, adnotata cu @Test:

```
@Test
public void testMethd_1() {
}
```

Aceasta metoda va apela la randul ei, metoda pe care vrem sa o testam (asa numitul business logic)

In interiorul metodei se efectueaza de obicei urmatorii pasi:

1. **Setup**: Se seteaza preconditiile, de ex. se initializeaza parametrii, se precalculeaza valoarea asteptata, se instantiaza obiecte, etc.
2. **Act**: Se apeleaza codul ce se vrea testat si se inregistreaza valorile returnate
3. **Assess**: Se compara valoarea returnata cu valoarea precalculata. Pasul 3 se foloseste de o metoda standard a bibliotecii TestNG, metoda numita assertEquals. Aceasta are 3(sau 4) parametri:

```
public static void assertEquals(int  actual, int  expected,
                                java.lang.String  message)
```

Primul parametru este valoarea returnata de codul ce am vrut sa il testam, al doilea valoarea precalculata iar al 3-lea este un mesaj de eroare personalizat in caz ca ceva nu a mers cum trebuie.

Metoda assertEquals va comunica cu pachetul TestNG. Acesta ne va reporta frumos cate teste au trecut, iar pentru cele care au esuat, ne va da mesaj cu valorile actual/expected si cu mesajul de eroare scris de noi.

Putem scrie cate metode de test vrem, intr-o clasa din directorul sursa *test*. De obicei, clasele de test sunt denumite dupa clasele pe care le testeaza, plus cuvantul Test. (ex MainTest pentru Main)

Daca codul de setup este mai lung, acesta poate fi izolat intr-o metoda separata care sa fie apelata manual sau automat inainte de fiecare test. Pentru a fi apelata automat, folosim adnotarea @BeforeTest.

Mai multe despre teste, in documentatia de la sfarsitul temei.

3 Practica

Continuati lucrul din laboratorul 3 sau descarcati laboratorul 4. Atentie, laboratorul 4 NU contine codul ce trebuia implementat la laboratorul 3.

3.1 Polinom de ordin general

Vom rescrie functia Func astfel incat sa accepte oricati coeficienti (dar cel putin 2). Coeficientii vor fi pastrati intr-un tabel. Gradul polinomului poate fi dedus din lungimea tabelului.

1. Folosind tabele, modificati membrii si constructorul. Daca lungimea vectorului de intrare este mai mica de 2, creati implicit functia f(x) = x
2. Modificati codul de la getFuncValue si getDerivValue astfel incat sa functioneze cu noile specificatii.
3. Adaugati o metoda numita toString() la clasa Func, metoda ce genereaza reprezentarea eleganta a functiei (ex: ”f(x)=2x^2-4x+1”). Folositi aceasta functie cand afisati radacinile (exemplu de mesaj „Pentru f(x)=.... radacina este 3”)

Mare atentie! Metoda toString() returneaza un string! NU afiseaza nimic pe ecran!

3.2 Scrierea de teste pentru polinom de ordin general

Scrieti teste pentru a va asigura ca ati implementat corect modificarile. Deocamdata testam doar calculul valorii si a derivatei in mai multe puncte.

Testam pentru polinomul x^3-x^2+2. Derivata acestuia este 3x^2-2x

Pentru a adauga un test, click dreapta pe numele clasei Func din Package Explorer si alegeti TestNG-> Create TestNG class. Dati click pe Finish.

Daca deja aveti in proiect clasa de test, modificati metodele existente astfel incat sa compileze si sa treaca. (s-ar putea sa nu compileze partea unde se aloca un polinom nou). Dupa care, treceti la adaugarea de teste.

Localizati functia adnotata cu @Test si schimbati-i numele in ceva mai sugestiv.

1. Scrieti partea de setup
    a. Creati un array cu parametrii polinomului
    b. Creati un obiect de tip Func cu parametrii de mai sus
    c. Calculati de mana, sau folosind expresii, valorile polinomului si a derivatei pentru x=1.5
2. Act:
    a. Apelati getFuncValue si getDerivValue si memorati valorile returnate
3. Assess:
    a. Folosind assertEquals verificati valorile obtinute, cu o precizie de 0.001

Observatie importanta: Testul trebuie sa testeze un singur lucru, asa ca, in mod normal, ati fi avut 2 teste, unul pentru valoarea polinomului si unul pentru valoarea derivatei.

Mai scrieti cateva teste pentru alte valori ale lui x.

Scrieti teste pentru polinom de gradul 2, 4, etc. (Macar inca 2 tipuri de polinoame, fiecare cu 2-3 valori)

Pe parcursul temelor vom invata si cum sa scriem mai eficient teste (sa evitam codul repetitiv) Pentru moment folositi cu incredere copy/paste

Rulati testele, trebuie sa treaca. Daca nu trec, intrati cu debug-ul si observati care sunt problemele.

3.3 Operarea cu obiecte

In clasa Main04, comentati pe moment codul din functia kbdRead() daca inca exista acolo.

In metoda main() a clasei Main04 (sau Main03):

Creati un tabel de cateva elemente, de tip Func. (denumiti-l funcTable) Populati-l cu cateva functii. In array-ul de mai sus (funcTable), faceti astfel incat indexul 0 si 1 sa mearga la ACELASI obiect.

Creati un tabel dobule cu valori de start.

Creati si apelati o metoda care accepta 2 array-uri (unul de tip Func si unul double) si returneaza un array de radacini pentru fiecare functie.

3.4 Modifcarea obiectelor

La clasa Func adaugati o metoda getter ce returneaza array-ul in care sunt stocati coeficientii.

Scrieti o metoda care ia un tabel de polinoame si afiseaza cate un polinom pe fiecare linie

Observam comportamentul referintelor descris la curs:

Modificati cativa coeficienti ai primului element din tabelul de polinoame, funcTable[0]

Afisati din nou polinoamele.

Observati ca s-a modificat si al 2-lea polinom!

3.5 Avansat:

(re)Faceti o metoda kbdRead() care sa citeasca n valori de la tastatura si sa creeze un polinom.

4 Evaluare

Tema se considera incheiata cand

1. Ati refacut clasa Func conform specificatiilor
4. Ati efectuat operatiile cu obiecte
5. Ati efectuat toti pasii de la punctul 3, cu accent pe teste
6. Toate testele sunt verzi.

5 Referinte:

http://www.mkyong.com/tutorials/testng-tutorials/
http://testng.org/doc/documentation-main.html
Cateva tutoriale despre testing:

http://www.javalobby.org/articles/testng/
http://tutorials.jenkov.com/java-unit-testing/simple-test.html
http://www.vogella.com/tutorials/JUnit/article.html

http://en.wikipedia.org/wiki/Extreme_programming
http://www.extremeprogramming.org/