Tema laborator 2
Variabile, instructiuni de control.

In acest laborator veti scrie cod Java simplu. Majoritatea instructiunilor de baza sunt ca si in C, deci, se presupun cunoscute. Veti citi primele 3 laboroatoare ca sa vedeti deosebirile fata de C si sa va aduceti aminte cum functioneaza variabilele si instructiunile de baza.

Deasemenea luati cunostinta cu instrumentele de baza de testare si depanare. Pentru primele laboratoare, veti primi testele scrise si voi doar le veti rula.

Cu depanarea trebuie sa va obisnuiti din start.

1 Materiale aditionale ce trebuie studiate

Cititi si intelegeti (de pe : http://users.utcluj.ro/~jim/OOPR/lab-announce.html)
Laborator 2: TipuriPrimitive+IOSimpla (Fara capitoul 4)
Laborator 3: Variabile+Expresii
Laborator 4: InstructiuniDeControl+ClaseSimple (Fara capitolul 4)

2 Teorie

In acest laborator veti scrie un cod ce calculeaza radacina patrata, folosind metoda Newton (vezi referinta spre wikipedia).

Pe scurt, daca vreau sa gasesc sqrt(S) trebuie sa fac pasii:

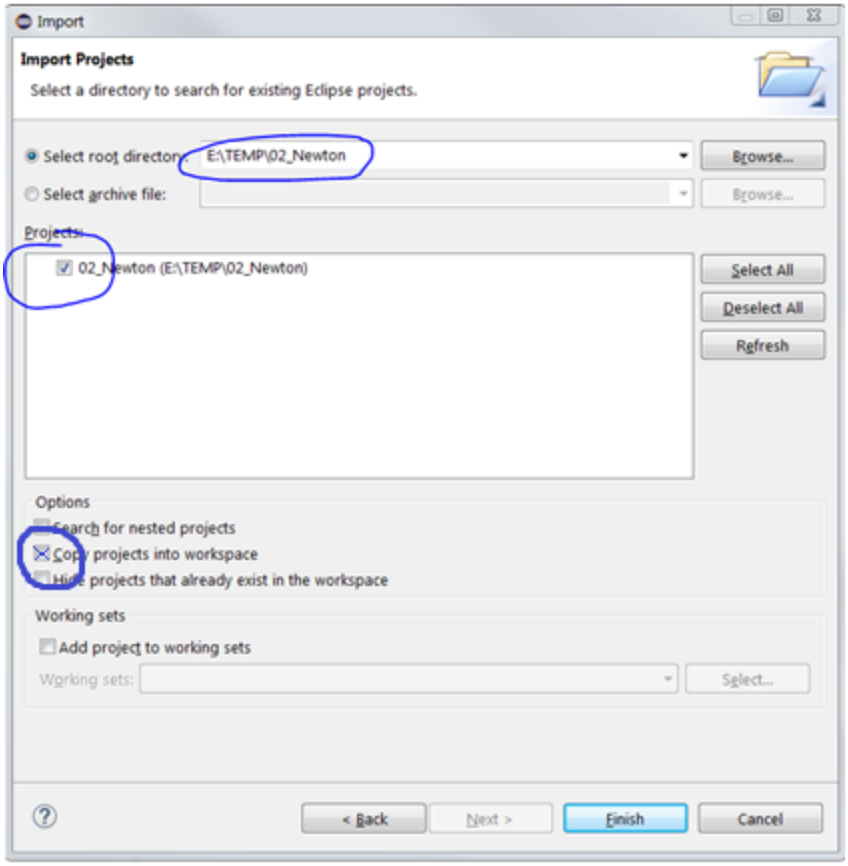
1) Se seteaza x=1
2) Se seteaza prag = 0.01
    a. Se calculeaza x1 = 1/2 (x+ S/x)
    b. Se verifica daca |x1 – x| < prag
        i. Daca da, se iese
    c. Se seteaza x = x1
    d. Se repeta pasul 2.a
3) Se returneaza valoarea lui x

Valoarea |p| reprezinta modulul lui p.

3 Practica

Descarcati si importati proiectul.

1. Descarcati si despachetati arhiva laboratorului undeva pe HDD (Ex. D:\Temp) Se va crea un director cu numele laboratorului (Ex D:\Temp\02_Newton)
2. In Eclipse dati File->Import si selectati General/Existing projects into Workspace
3. La Root directory selectati directorul laboratorului (Ex D:\Temp\02_Newton)
4. Selectati proiectul, daca nu este deja selectat.
5. Bifati “Copy projects into workspace
6. Dati click pe Finish.
7. Proiectul va aparea in „Package explorer”



Deschideti fisierul MainL02 si localizati zona unde trebuie sa scrieti cod.

MainL02.java | MainL02Test.java

```
1  package ro.utcluj.poo.lab02;
2
3  public class MainL02 {
4      /**
5       * Metoda calculeaza radical din S folosind metoda Newton
6       * @param S valoarea pentru care se calculeaza radicalul
7       * @return valoarea radicalului
8       */
9      public double getSqrt(double S){
10         double x, x1;
11         double prag;
12         x = 1;
13         prag = 0.01;
14         //AICI SCRIETI CODUL CARE REZOLVA PROBLEMA
15
16
17
18         //*************
19         return x;
20     }
21
22     public static void main(String[] args) {
```

Rulati testele (vezi cap urmator), rulati programul sa vedeti cum se comporta in acest moment.

Implementati codul folosind variabile, instructiuni de atribuire, instructiuni de control, etc. Eclipse subliniaza cu rosu erorile de compilare. Acestea se pot observa si in fereastra Problems, afisata sub cod. Cititi cu atentie mesajele de eroare si corectati.

Odata ce programul compileaza (Nu mai apar erori la Problems) puteti rula.

Ca sa va verificati, rulati din nou testele. In principiu ar trebui sa treceti usor primele doua teste. Testul al 3-lea calculeaza valoarea cu o precizie prea mica (gaseste 0.01777 in loc de 0.01).

4 Depanarea programului

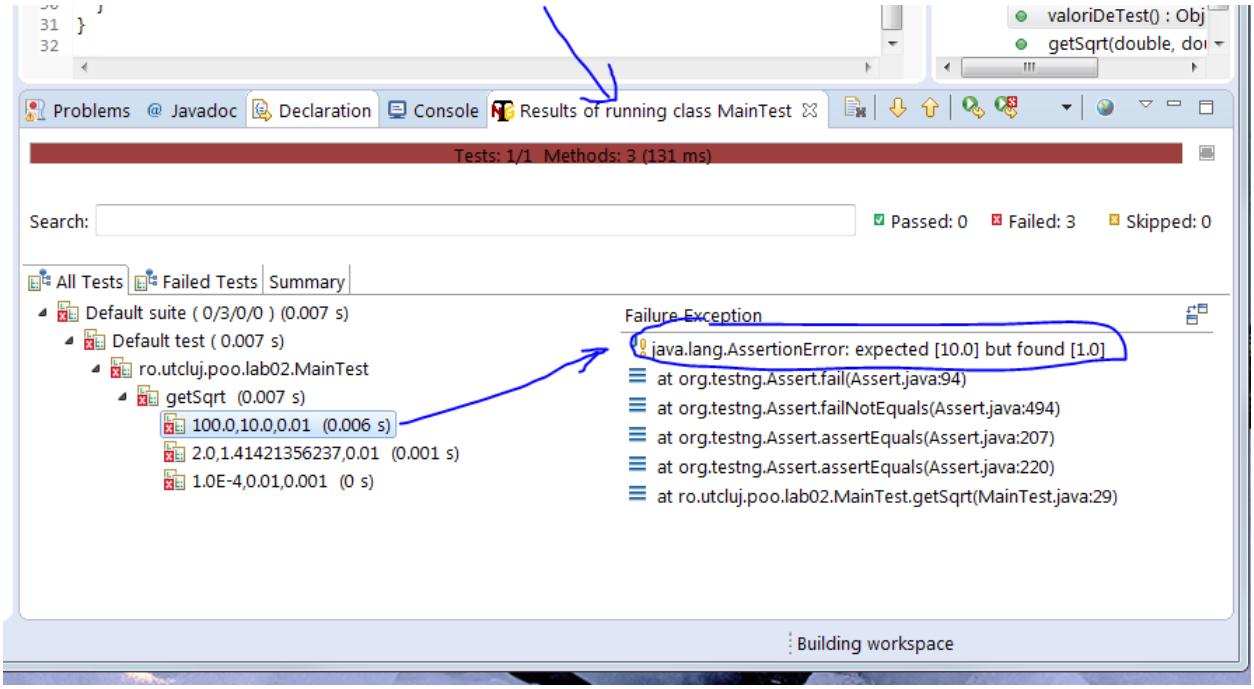
4.1 Rularea testelor

Deschideti fisierul MainL02Test si faceti click pe Run.

Alta alternativa e sa faceti click dreapta pe fisier, din Package explorer si sa selectati Run As-> TestNG Test.

De obicei, gasiti rularile recente facand click pe sagetuta de langa butonul de run.

Dupa ce ati rulat testele, deschideti „Results of running class MainTest” si observati rezultatul.



Cu rosu, apar testele care nu au trecut. Facand click pe test, vi se da detalii suplimentare. In acest caz, testul astepta raspunsul 10, dar a primit 1.

Puteti repeta testele de cate ori vreti. Cu click dreapta si alegand Run, puteti rula doar un anumit test.

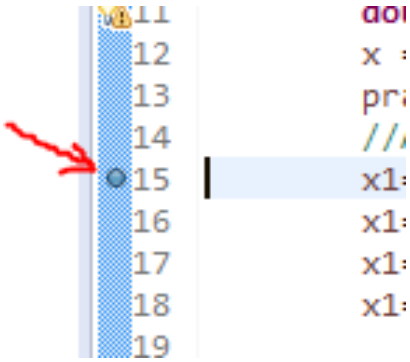
4.2 Cum se foloseste debugger-ul

Depanarea unui program presupune executarea linie cu linie a unei bucati de cod, cu monitorizarea variabilelor si a liniilor ce se executa. Principiile de baza va sunt familiare de la materia Programarea Calculatoarelor, aici vi se arata doar cateva detalii specifice Eclipse.

Este foarte important sa va insusiti depanarea!

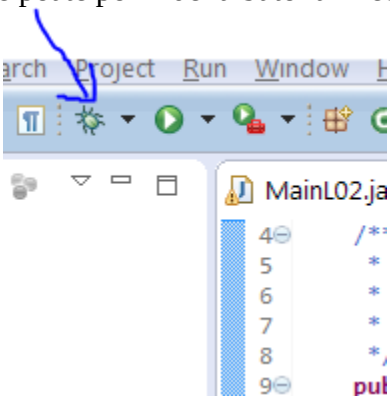
Sa presupunem ca ati implementat un cod, si doar anumite teste nu trec. Puteti sa faceti depanare doar pe acel test, urmand urmatorii pasi:

8. Setati punctul de intrerupere (Breakpoint) facand dublu click pe numarul liniei (ex linia 15):

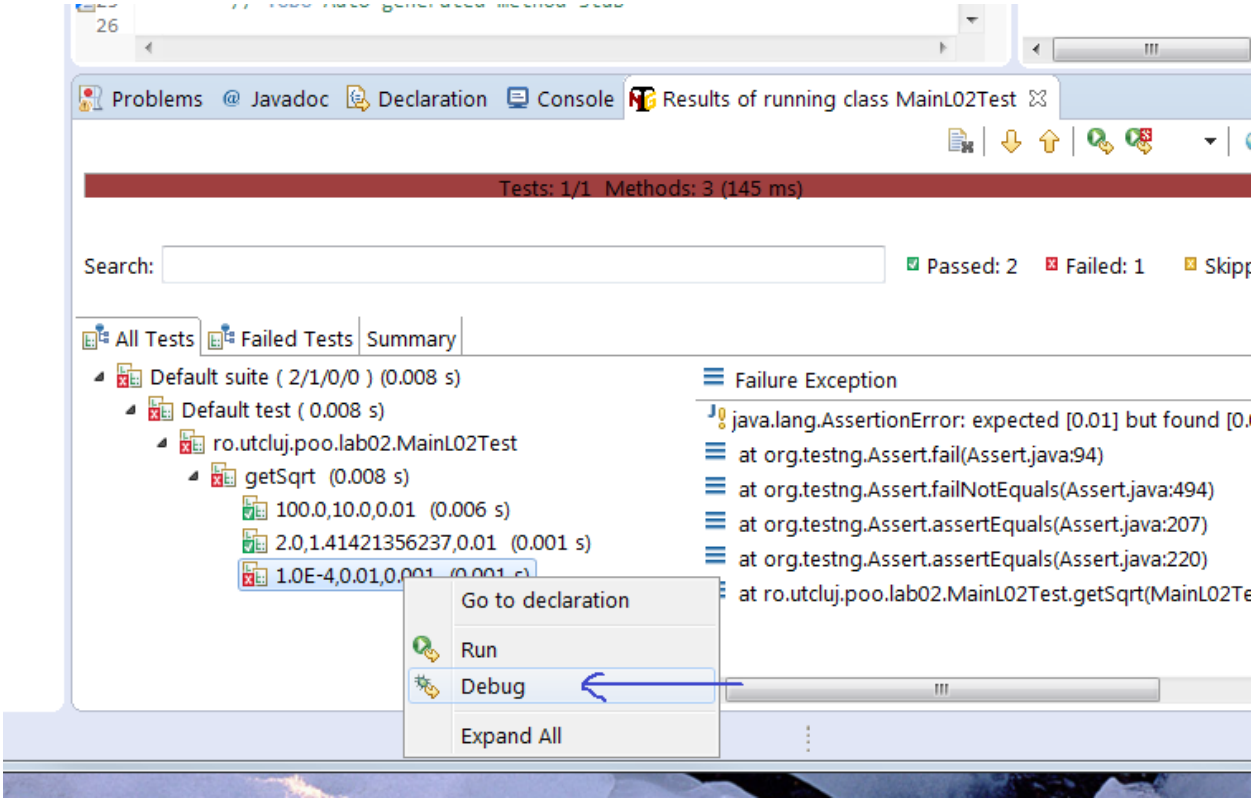


Va aparea un punct albastru, semnificand setarea breakpoint-ului.

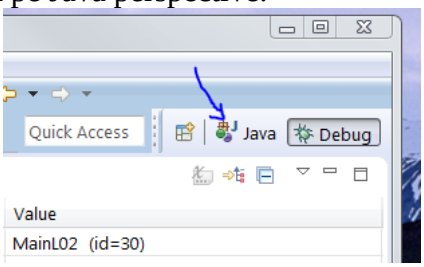
9. Porniti aplicatia/testul in modul „Debug”:
   a. Aplicatia se poate porni de la butonul Debug (situat langa Run):



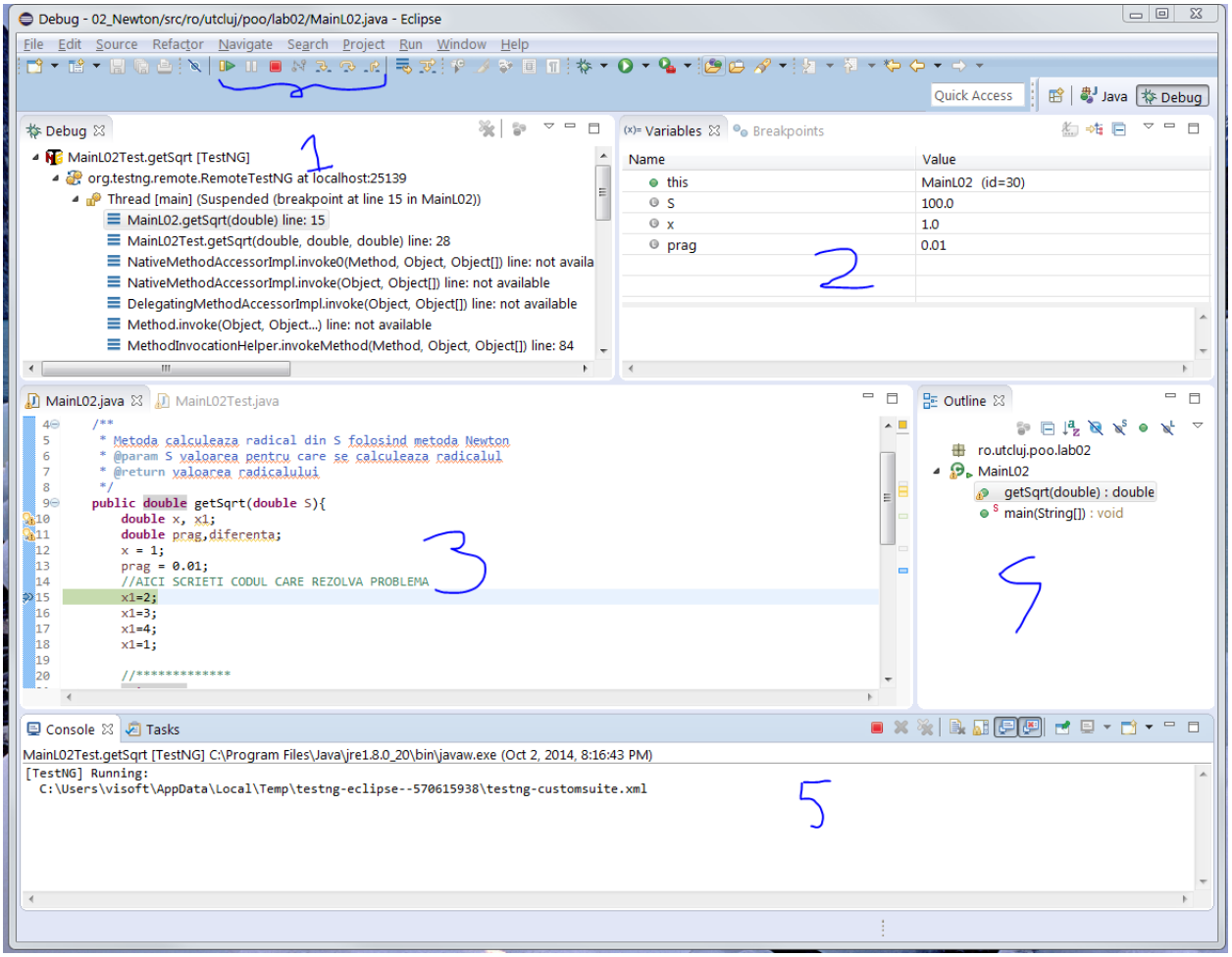
   b. A 2-a varianta (cea recomandata) este sa porniti strict testul care esueaza in modul debug: Click dreapta pe test si alegeti Debug:



10. Aplicatia se va opri in dreptul primei linii marcate cu breakpoint (puteti seta mai multe astfel de puncte de intrerupere) si asteapta inputul vostru. Deasemenea, ceva foarte important, se schimba si modul de afisare a informatiei pe ecran. Daca va place acest mod, puteti sa il pastrati. Daca nu, puteti reveni oricand la modul anterior, facand click pe Java perspective:



Revenind la depanare, mai jos aveti perspectiva:



Semnificatie:

1: Stiva de apeluri. Puteti vedea cum s-a ajuns la linia curenta

2: Lista de variabile. Aici puteti vedea variabilele locale declarate in momentul executiei liniei. Puteti modifica valorile fiecarei variabile. Daca vreti sa vedeti si alte variabile (sau expresii) puteti face selectand Run->Watch si adaugand variabile in aceasta fereastra. (Alternativ, click dreapta in zona 2 si selectati Watch)

3: Zona de cod. Linia curenta este marcata cu o sageata si subliniata cu verde. Observati ca s-a oprit in dreptul liniei.

4: Structura proiectului. Util daca vreti sa navigati in alta functie (ex. sa setati un nou breakpoint)

5: Zona de consola.

11. Depanarea programului: Folosind Step Into (F5), Step Over (F6), Step Return (F7), Resume (F8) puteti rula pas cu pas, iesi din metoda, sau executa programul pana la urmatorul punct de intrerupere. Cele mai utile prescurtari sunt F6 si F8.

Odata ce inaintati linie cu linie, in fereastra Variables, variabilele ce se modifica sunt ingalbenite.

5 Rezolvarea ultimului test

Executand pas cu pas algoritmul de la capitolul 2 observati la punctul 2.b validarea prematura a conditiei |x1-x| < prag mai ales daca valoarea lui S este mult mai mica decat pragul.

Puteti sa va ganditi la alternative de oprire a executiei buclei de calcul, alternative ce sa mearga atat pe numere mari (gen sqrt(100) == 10) dar si pe numere foarte mici (sqrt(0.0001) == 0.01).

O solutie simpla ar fi sa setam pragul ca fiind un procent (ex 0.01) din valoarea lui S. Asta ar insemna sa verificam daca |x1-x| < prag*S. Ruland din nou testul, se observa ca trece! Pentru valoarea de intrare 0.0001, valoarea de iesire este identificata corect ca fiind 0.01. Dar oare nu am stricat ceva modificand conditia de oprire?

Avantajul major al testelor este ca le putem rula oricand, de cate ori vrem! Ruland intreg fisierul de test, observam ca primul test (input = 100, expected output = 10) esueaza! Inseamna ca trebuie sa mai lucram la conditia de oprire.

O alta sugestie ar fi sa selectam pragul conditionat. Pentru valori supraunitare, lasam pragul implicit, iar pentru valori subunitare, folosim valoarea prag*S. Cream o noua variabila numita prag2, de tip double, la care ii setam conditionat valoarea.

Ruland din nou testele, observam verde!

Observatie: In practica, este important sa rezolvam problema intr-un mod acceptabil, chiar daca nu este cel mai elegant sau eficient mod. In exemplul de fata, cerinta este de a face cele trei teste sa treaca.

6 Evaluare

Tema se considera incheiata cand toate testele sunt verzi.

7 Referinte:

http://en.wikipedia.org/wiki/Methods_of_computing_square_roots