Tema 1 - Proiectarea si dezvoltarea unui prototip de evaluare a functiilor de o singura variabila reala

Mostenire.Clase abstracte si interfete.Structurare in package-uri. JavaDoc

-Recapitulare si completare Java-

Exercitiu - Functii de o singura variabila reala (Evaluare si derivare simbolica)

Scopul este de a construi o aplicatie care sa trateze functiile de o singura variabila reala.
De exemplu f(x)= 3x + sin(x).
Diversi operatori aritmetici si functii pot fi folositi în definirea acestor functii : +,*, sin, ln, exp,etc.
Forma interna utilizata pentru a codifica functiile ce urmeaza sa fie manipulate va fi o forma arborescenta, identificabila prin radacina sa.

Aplicatia ce urmeaza sa fie dezvoltata va avea urmatoarele functionalitati :
a) Evaluarea functiei pentru o anumita valoare a argumentului.
b) Calcularea derivatei pentru o anumita valoare a argumentului ;
c) Calcularea derivatei simbolice;

Pentru aceasta aplicatie generati documentatia JavaDoc. Structurati clasele aplicatiei in package-uri in functie de rolul acestora.

Indicatii de rezolvare

1. **Intelegerea problemei**

Rolul acestei aplicatii este de :

- a evalua o functie pentru o anumita valoare ;
- a deriva la nivel simbolic o functie ;
- a calcula derivata functiei pentru o anumita valoare data ;

De exemplu :

Fie functia f(x)=sin(x)+2x si x=2.

Aplicatia va trebui :

- sa afiseze f(2)=4+sin(2) ;
- sa afiseze derivata simbolica, adica f'(x)=cos(x)+2 .
- sa calculeze valoarea derivatei pentru x=2 , adica f'(2)=cos(2)+2.

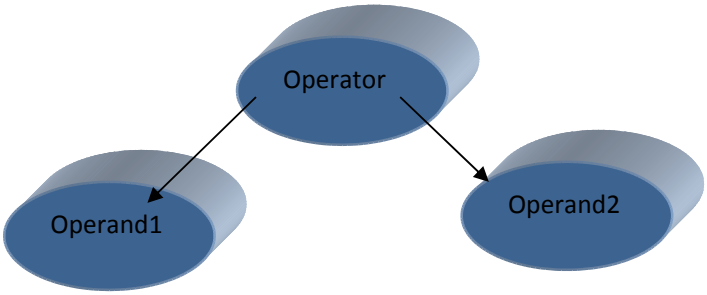
2 . Alegerea modelului de reprezentare a functiei

Aceasta aplicatie lucreaza cu functii. Deci va fi necesara constructia unei clase numita Functie. Problema fundamentala care se pune este cum va fi memorata aceasta functie de catre aplicatie si cum se va opera cu ea.In primul rand functia va fi retinuta de catre aplicatie intr-un String. Aceasta modalitate de memorare nu este insa optima dat fiind faprtul ca nu se poate opera cu acaesta structura de date pentru a obtine drivari simbolice si calcule numerice. Va trebui asadar gasit un alt model care sa permita lucrul facil cu functii.

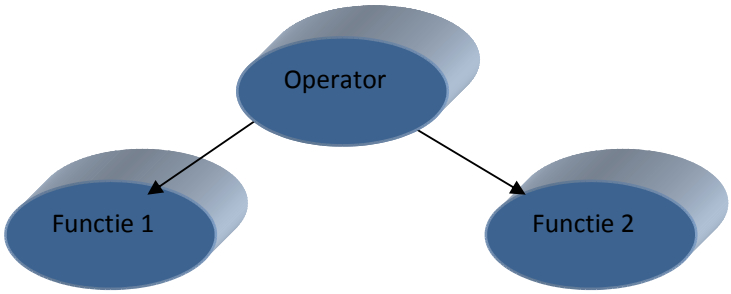
O functie este compusa din :

- operatori : +,-,*, / .
- functii elementare : functia constanta,functia identica (f: R->R, f(x)=x), functia putere,functia radical, functia logaritm, functia exponentiala, functiile trigonometrice (sin,cos,tg,ctg,arcsin,arccos,arctg,arcotg).

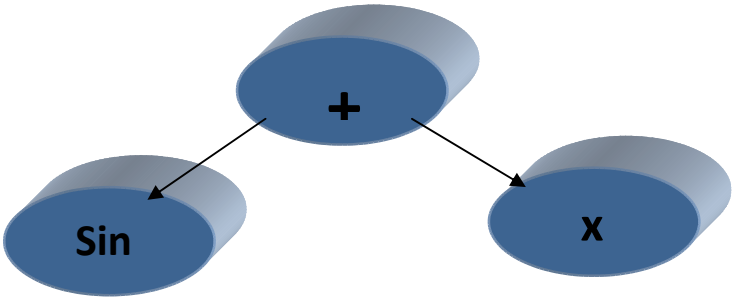
Un operator are intotdeauna doi operanzi. Aceasta se poate reprezenta in diferite moduri,folosind de exemplu vectori, liste,arbori,etc.Cel mai intuitiv este modul grafic urmator :



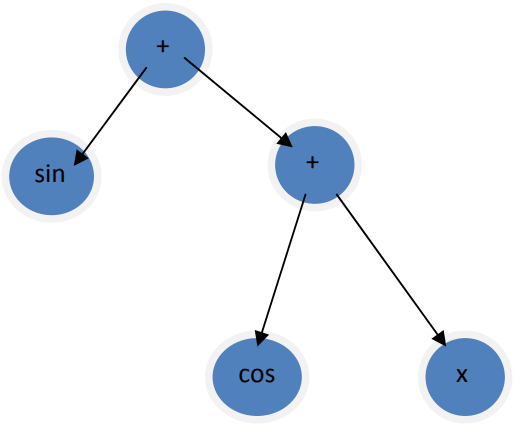
Un operand nu poate sa fie decat o functie.Deci o reprezentare mai concreta ar fi :



De exemplu pentru f(x)=sin(x) + x o posibila reprezentare ar fi :

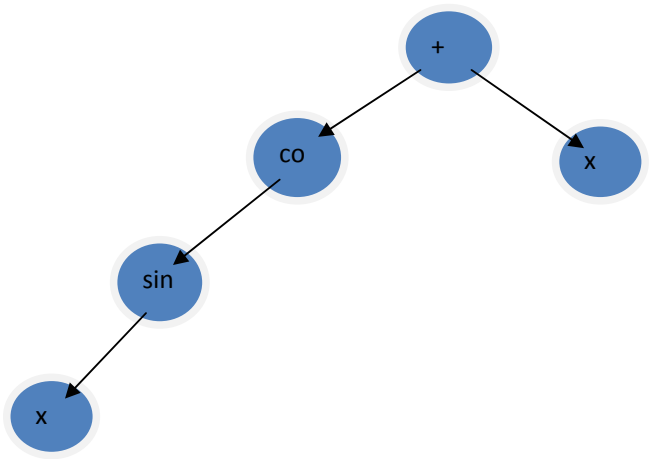


Pentru functia f(x)=sin(x)+cos(x)+x,reprezentarea grafica va fi :



Deci modelul prezentat si potrivit pentru a memora functii este de **arbore binar**,dat fiind faptul ca toti operatorii sunt binari.

O alta problema care se pune este legata de functiile compuse. De exemplu f(x)=cos(sin(x))+x. Solutia propusa este ca orice functie sa aibe ca si argument fie functia identica fie o alta functie. Drept consecinta toate frunzele arborelui binar vor fi functia identica, f(x)=x . Conform cu ultima afirmatie,functia f(x)=cos(sin(x))+x va avea reprezentarea arborescenta :



În concluzie putem spune ca structura de date folosita pentru a repreznta o functie algebrica este de arbore binar,unde frunzele arborelui reprezinta functia identica, nodurile cu doi fii reprezinta operatori aritmetici elementari(+,-,*,/) iar nodurile cu un singur fiu functii matematice elementare (putere,logaritm,sin, etc.).

Exercitiu :

Reprezentati sub forma arboresecenta urmatoarele functii :

f(x)= ln(sin(x))+ exp(x)-3

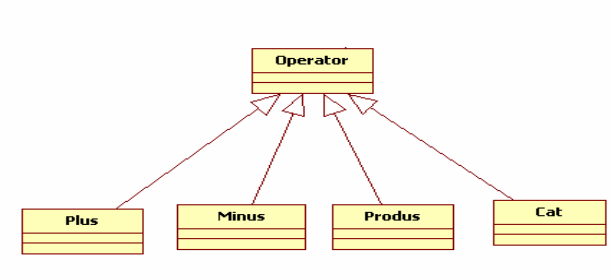
f(x)=sin(x)cos(x)+exp(tg(x))-exp(x)log(x)arctg(x)

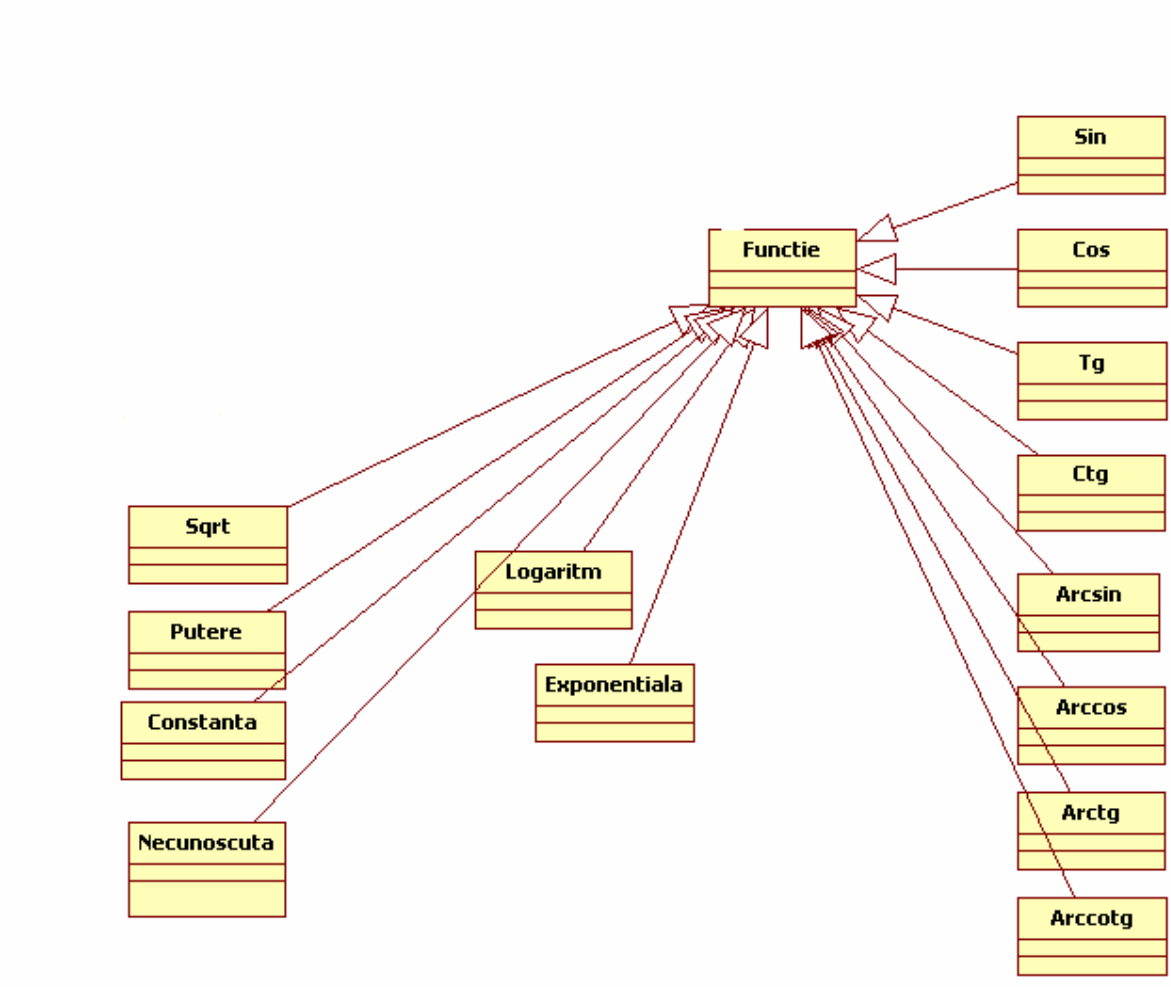
f(x)=arcsin(x) arccos(x) arctg(x) + sqrt(x+2.3)

3. Modelarea orientata obiect a aplicatiei

Deci aplicatia va trebui sa modeleze conceptele prezentate mai sus : operatori si functii elementare. Conform principiilor de proiectare orientate obiect pentru fiecare concept se recomanda definirea unei clase. Asadar vom avea urmatoarele clase : Plus, Minus, Produs, Cat, Constanta, Necunoscuta (clasa care reprezinta functia identica sau necunoscuta x), Putere, Radical, Logaritm, Exponentiala, Sin, Cos, Tg, Ctg, Arcsin,Arccos,Arctg,Arcotg, etc.

Dintre functiile de mai sus, o parte din ele le putem grupa în categoria Operatori si alta in categoria Functie. Deci putem avea o prima ierarhie de clase,reprezentata de urmatoarele doua figuri :





Atat operatorii cat si functiile sunt pentru structura arborescenta ce reprezinta functia noduri. Dat fiind faptul ca cele doua clase Operator si Functie reprezinta concepte care fac parte din aceeasi categorie putem cauta o

clasa comuna .Deci putem construi o clasa numita Nod,care va fi clasa parinte pentru clasele Operator si Functie. Deci o reprezentare grafica a tuturor claselor aplicatiei poate fi exemplificata de figura urmatoare :

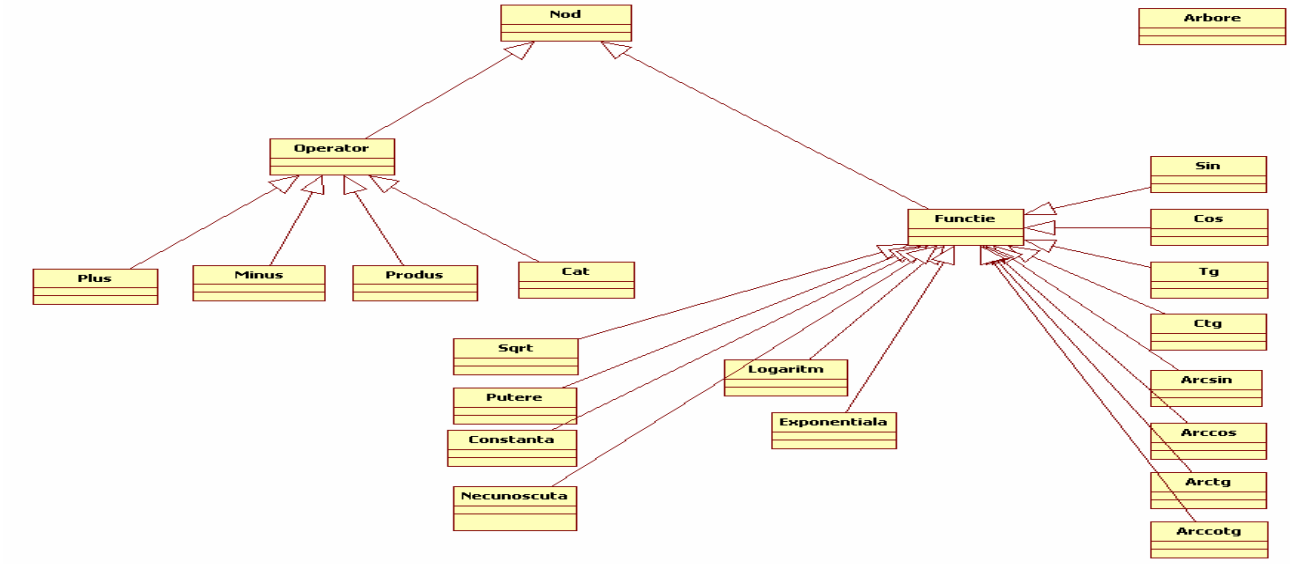
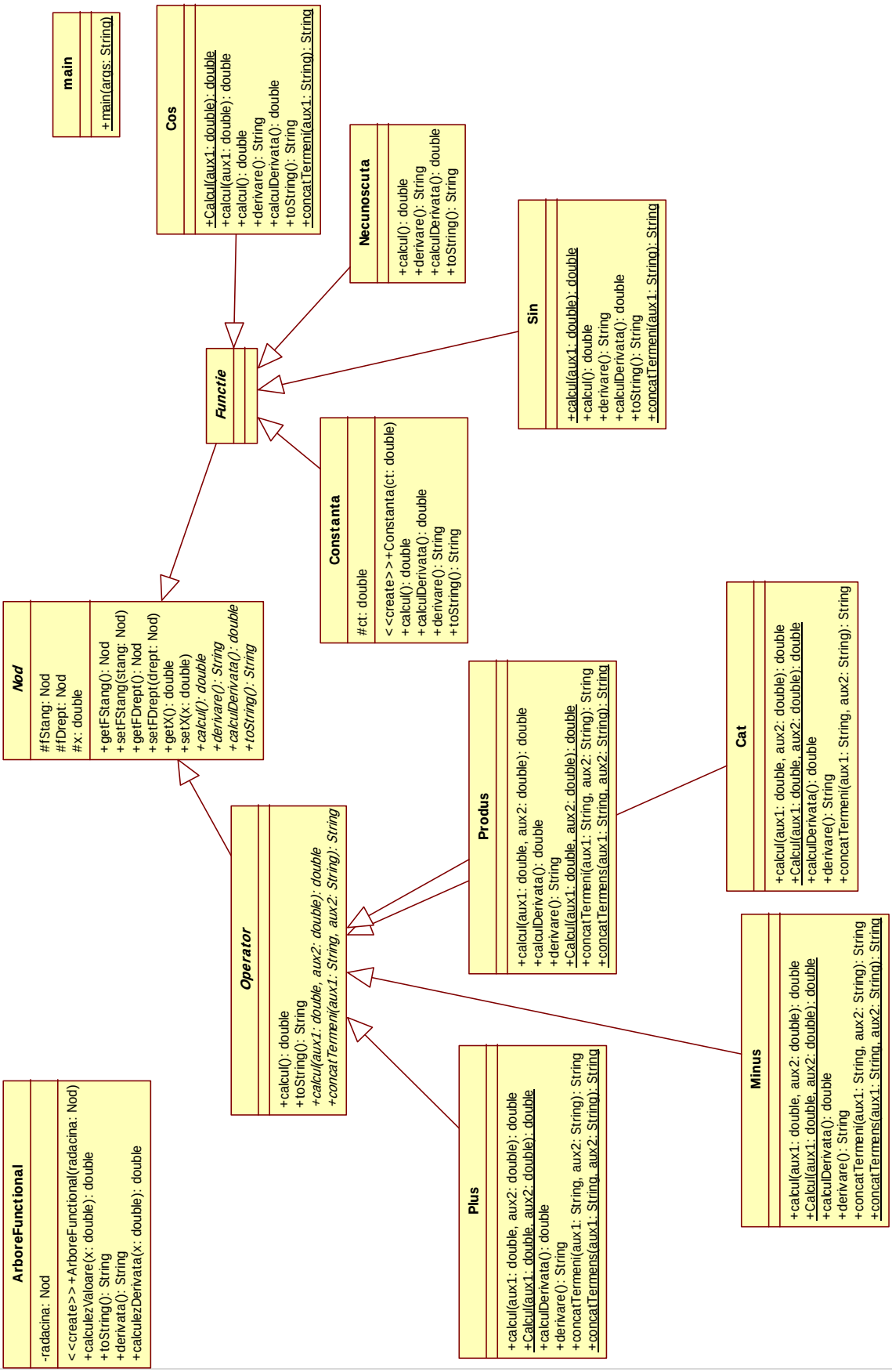


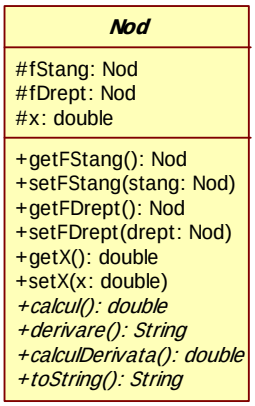
Diagrama de clasa a aplicatiei va fi urmatoarea :


main
+main(args: String)
Cos
+Calcul(aux1: double): double
+calcul(aux1: double): double
+calcul(): double
+derivare(): String
+calculDerivata(): double
+toString(): String
+concatTermeni(aux1: String): String
Necunoscuta
+calcul(): double
+derivare(): String
+calculDerivata(): double
+toString(): String
Sin
+calcul(aux1: double): double
+calcul(): double
+derivare(): String
+calculDerivata(): double
+toString(): String
+concatTermeni(aux1: String): String
Functie
Constanta
#ct: double
<<create>>+Constanta(ct: double)
+calcul(): double
+calculDerivata(): double
+derivare(): String
+toString(): String
Nod
#fStang: Nod
#fDrept: Nod
#x: double
+getFStang(): Nod
+setFStang(stang: Nod)
+getFDrept(): Nod
+setFDrept(drept: Nod)
+getX(): double
+setX(x: double)
+calcul(): double
+derivare(): String
+calculDerivata(): double
+toString(): String
Operator
+calcul(): double
+toString(): String
+calcul(aux1: double, aux2: double): double
+concatTermeni(aux1: String, aux2: String): String
Produs
+calcul(aux1: double, aux2: double): double
+calculDerivata(): double
+derivare(): String
+Calcul(aux1: double, aux2: double): double
+concatTermeni(aux1: String, aux2: String): String
+concatTermens(aux1: String, aux2: String): String
Cat
+calcul(aux1: double, aux2: double): double
+Calcul(aux1: double, aux2: double): double
+calculDerivata(): double
+derivare(): String
+concatTermeni(aux1: String, aux2: String): String
ArboreFunctional
-radacina: Nod
<<create>>+ArboreFunctional(radacina: Nod)
+calculezValoare(x: double): double
+toString(): String
+derivata(): String
+calculezDerivata(x: double): double
Plus
+calcul(aux1: double, aux2: double): double
+Calcul(aux1: double, aux2: double): double
+calculDerivata(): double
+derivare(): String
+concatTermeni(aux1: String, aux2: String): String
+concatTermens(aux1: String, aux2: String): String
Minus
+calcul(aux1: double, aux2: double): double
+Calcul(aux1: double, aux2: double): double
+calculDerivata(): double
+derivare(): String
+concatTermeni(aux1: String, aux2: String): String
+concatTermens(aux1: String, aux2: String): String

4. Implementarea ierarhiei de clase

In continuare vom prezenta pe rand fiecare clasa a aplicatiei :

a. Clasa Nod



- Atributele clasei sunt de tipul protected. Aceasta inseamna ca ele sunt accesibile doar metodelor interne ale clasei Nod precum si tuturor metodelor claselor derivate. Încercati sa le declarati de tipul private. Atributele de tip privat pot fi accesate doar de catre metodele clasei respective, nu insa si de catre clasele fiu. In cazul de fata daca atributul fStang va fi privat, pentru ca in clasa Plus sa fie accesat acest atribut :
  - o instructiune de tipul fStang.toString() va esua ;
  - o instructiune de tipul getFstang().toString() va afisa cu succes;

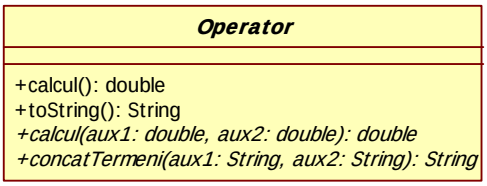
- Generati automat cu ajutorul mediului IDE Eclipse metodele get si set pentru toate atributele clasei. (Source->Generate Getters and Setters).

- Metodele calcul(), derivare(),calculDerivata(),toString() vor fi metode abstracte. O metoda abstracta se declara cu ajutorul cuvnatului cheie abstract in fata ei si nu au corp. Un exemplu de metoda abstracta este :

```
public abstract  double calcul();
```

Atentie : o clasa care are cel putin o metoda abstracta trebuie declarata abstracta.

b. Clasa Operator



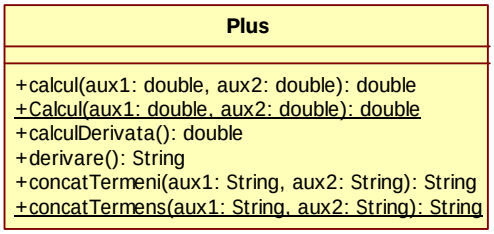
Clasa Operator mosteneste clasa Nod. Aceasta se exprima în Java cu ajutorul cuvantului cheie extends.

```
public abstract class Operator extends Nod{
…
```

- Metoda calcul se va ocupa de calculele care trebuie sa le faca un operator. Toti operatorii au întotdeauna doi operanzi. Asadar aceasta metoda va verifica daca cei doi operanzi exista, urmand ca apoi sa paseze metodei abstracte calcul(double,double) sarcina de a efectua calculele efective. Testarea existentei celor doi operanzi se face testand daca fiul stang,respectiv cel drept sunt diferiti de obiectul nul, adica null. In cazul in care unul dintre operanzi este nul se va iesi din program cu ajutorul instrctiunii System.exit(0). Metodei calcul(double,double) i se vor trimite ca si parametri rezultatele obtinute la calcul de catre fii,adica se va folosi un mecansim recursiv.
- Metoda toString() va prelua sirurile de caractere de la nodurile fiu stang respectiv fiu drept,si va concatena apoi cele doua siruri cu ajutorul metodei abstracte concatTermeni(String,String).Pentru a obtine sirul de caractere corespunzator unui nod fiu se va apela metoda toString() a acestuia.
- Metodele calcul si concatTermeni sunt metode abstracte. Ele se declara cu ajutorul cuvantului cheie abstract in fata si nu au corp.

c. Clasele Plus,Minus, Produs, Cat

Aceste clase mostenesc clasa abstracta Operator.Dat fiind faptul ca aceste clase sunt clase concrete si nu abstracte ele vor trebui sa implementeze toate metodele clasei abstracte Operator respectiv Nod.

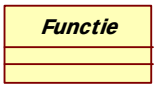



```
public class Plus extends Operator {


    public double calcul(double aux1, double aux2)
    {
        return aux1 + aux2;
    }

    public static double Calcul(double aux1, double aux2)
    {
        return aux1 + aux2;
    }

    public  double calculDerivata()
    {
        if (fStang==null || fDrept==null)
            System.exit(1);

        return calcul(fStang.calculDerivata(), fDrept.calculDerivata());
    }

    public  String derivare()
    {
         if (fStang==null || fDrept==null)
            System.exit(1);

        return concatTermeni(fStang.derivare(), fDrept.derivare());
    }
```

```
    public String concatTermeni(String aux1, String aux2)
      {
      String aux;
          if (aux1.compareTo("0") == 0 || aux2.compareTo("0") == 0)
          //if (!strcmp(aux1, "0") && !strcmp(aux2, "0"))
              aux = aux1;
          else if (aux1.compareTo("0") == 0) aux = aux2;
          else if (aux2.compareTo("0") == 0 ) aux = aux1;
          else
              aux = aux1 + "+" + aux2;
          return aux;
      }

      public static String concatTermens(String aux1, String aux2)
      {
          String aux;
          if (aux1.compareTo("0") == 0 || aux2.compareTo("0") == 0)
          //if (!strcmp(aux1, "0") && !strcmp(aux2, "0"))
              aux = aux1;
          else if (aux1.compareTo("0") == 0) aux = aux2;
          else if (aux2.compareTo("0") == 0 ) aux = aux1;
          else
              aux =  aux1 + "+" + aux2;
          return aux;
      }

}
```

În mod asemanator construiti clasele Minus, Produs si Cat.

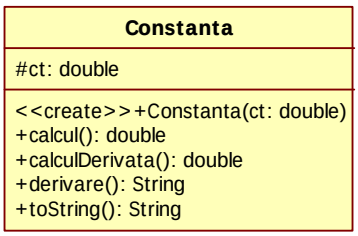
d. Clasa Functie




```
public abstract class Functie extends Nod {

 }
```

Pentru moment clasa Functie va avea corpul vid. Modificati corpul clasei Functie dupa ce veti scrie clasa Constanta.

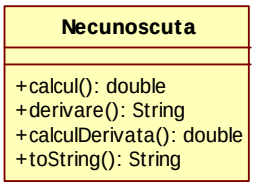
e. Clasa Constanta



Aceasta clasa reprezinta constantele care pot sa apara intr-o functie algebrica. De exemplu în f(x)=3+x apare constanta 3. Obiectele de tip constanta vor trebui sa memoreze valoarea constantei respective. Aceasta se va face cu ajutorul unui atribut numit ct.

- Constructorul nu va face decat initializarea campului ct pentru fiecare obiect în parte.
- Metoda calcul va returna valoarea constantei;
- Metoda calculDerivata va returna valoarea constantei derivate,adica 0;
- Metoda derivare va returna un String ce reprezinta derivata unei constante adica „0”;
- Metoda toString va returna un String ce va contine valoarea constantei adica ct. Atentie deoarece un double va trebui transformat în String.

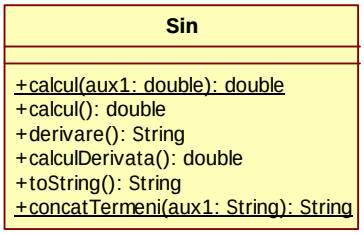
f. Clasa Necunoscuta



Clasa Nod are un atribut x de tip double. Acest atribut este folosit pentru a memora argumentul functiei folosit la diferite calcule non-simbolice. Clasa Necunoscuta face diferite calcule pornind de la aceasta valoare.

- Metoda calcul() va returna tocmai valoarea lui x;
- Metoda derivare() va returna un String ce va contine pe x’ adica „1”.
- Metoda calculDerivata() va returna x’ adica 1.
- Metoda toString() va returna un String ce reprezinta o necunoscuta, adica „x”.

g. Clasa Sin

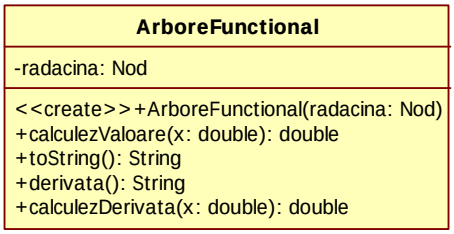



```
public class Sin extends Functie {

    public static double calcul(double aux1)
    {
        return Math.sin(aux1);
    }


    public double calcul()
    {
        if (fStang == null)
            System.exit(1);

        return calcul(fStang.calcul());
    }

    public String derivare()
    {
        if (fStang == null)
            System.exit(1);
```

```
        return Produs.concatTermens(fStang.derivare(),
                                    Cos.concatTermeni(fStang.toString()));
    }

    public double calculDerivata()
    {
        if (fStang == null)
            System.exit(1);

        return Produs.Calcul(fStang.calculDerivata(),
                Cos.Calcul(fStang.calcul()));
    }

    public String toString()
    {
        if (fStang == null)
            System.exit(1);

        return concatTermeni(fStang.toString());
    }

    public static String concatTermeni(String aux1)
    {
        String aux;
        int nrP;

        nrP = aux1.charAt(0) == '(' ? 0 : 2;

        aux = "sin";
        if (nrP != 0) aux += "(";
        aux += aux1;
        if (nrP != 0) aux += ")";

        return aux;
    }
}
```

În mod analog construiti clasele Cos, Putere, Radical, Logaritm, etc.

h.Clasa ArboreFunctional



- Un arbore are ca si trasatura predominanta radadina sa. Radacina memoreaza la randul ei doua referinte catre fiecare dintre clasele sale fiu. La fel fiaecare fiu memoreaza referinte catre fii sai si tot asa mai departe. Este la fel ca si in viata reala unde cel mai importanta pentru copac este radacina sa,crengile ramificandu-se apoi in mod aleatoriu fara a sti de la inceput incotro se vor orienta. Aceasta este înca un argument în plus pentru a motiva de ce un arbore este o structura dinamica de date.
  - Constructorul clasei va initializa nodul radacina al arborelui cu nodul pasat ca si parametru.

- Metoda calculezValoare(double) apeleaza metoda calcul( ) a nodului radacina. Înainte de a apela aceasta metoda se seteaza atributul x al nodului radacina cu valoarea transmisa ca si parametru. Modificati în mod corespunzator metoda setX(double) a clasei Nod astfel încât la setarea atributului x al unui nod sa se seteze cu aceeasi valoare si nodurile fiu (! Se va testa existenta nodului fiu si apoi se va seta valoarea acestuia).
- Metoda toString() apeleaza metoda toString( ) a nodului radacina.
- Metoda derivata( ) apeleaza metoda derivare( ) a nodului radacina.
- Metoda calculezDerivata(double) va calcula valoarea derivatei pentru argumentul transmis ca si parametru. Se seteaza corespunzator atributul x al radacinii si apoi se va apela metoda calculDerivata( ) pentru radacina.

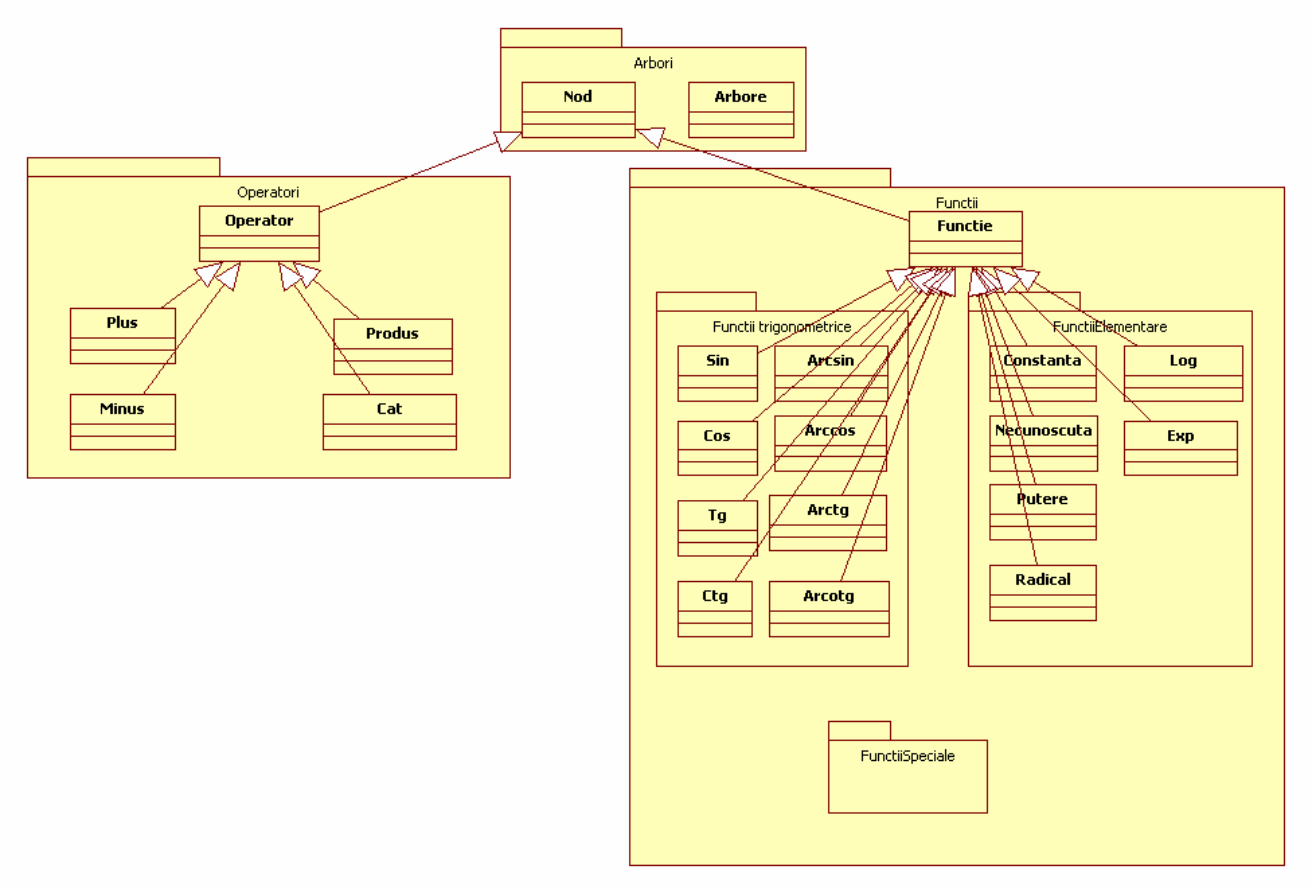
i. Clasa Main

```
public class main {

/**
 * @param args
 */
public static void main(String[] args) {
      // TODO Auto-generated method stub
 Nod n=new Necunoscuta();
 Nod sin=new Sin();
 Nod cos=new Cos();
 Nod xn=new Necunoscuta();
 Nod cat=new Cat();

 cat.setFStang(n);
 cat.setFDrept(sin);
 sin.setFStang(cos);
 cos.setFStang(xn);

 ArboreFunctional a=new ArboreFunctional(cat);

 System.out.println(a.derivata());
}

}
```

5. Structurarare in package-uri. Modificatorul package

Aceasta aplicatie are un numar mare de clase.Orice aplicatie Java utilizeaza un numar impresionant de clase, aceasta fiind consecinta programarii orientate obiecte : clase mici si usor de inteles si de modificat, flexibilitatea arhitecturii aplicatiei pentru a raspunde la functionalitati noi. Numarul de clase în general însa creste direct proportional cu complexitatea aplicatiei. Problema care se pune este de a structura clasele astfel încat sa fie mai usor de gestionat si de manipulat. De aici s-a nascut ideea de pachet, care contine mai multe clase Java. Unul dintre principiile programarii orientate obiect este modularitatea, principiu care se materializeaza în limbajul Java prin notiunea de pachet-package.

Structurati clasele aplicatiei asa cum este prezentat în figura urmatoare. Înainte de a aranja clasele asa cum este indicat în figura urmatoare declarati clasele Plus, Minus, Produs, Cat, Sin, Cos, Tg, Ctg, Arcsin, Arccos, Arctg, Arccotg, Constanta, Necunoscuta, Putere, Radical, Log, Exp ca si clase finale (de ce ?) .



6. Generarea automata a documentatiei JavaDoc

Comentati toate clasele aplicatiei utilizand comentarii JavaDOc. Specificati in comentariile dumneavoastra semnificatia atributelor de clasa, iar pentru metode explicati rolul ei, ce returneaza si de asemeni care este semnificatia parametrilor primiti.

Aceasta tema trebuie efectuata in binom (grupa de 2 persoane).

Termen de predare :