POO – Laborator 2

Introducere in programarea orientata spre obiecte (POO). Obiecte si clase. Incapsularea

2.1. Descrierea laboratorului

In aceasta lucrare de laborator vor fi acoperite urmatoarele probleme:

- Utilizarea mediului de dezvoltare integrat (IDE – *Integrated Development Environment*) BlueJ (tutorial, adaptare in limba romana) – **link extern**

- **Obiecte si clase (Java)**
  - Introducere in obiecte si clase Java (adaptare dupa R. Baldwin) – **link extern**
  - Introducere in Java, Introducere in OO cu Java, Clase Java de biblioteca – **linkuri externe**
  - **Crearea obiectelor si invocarea metodelor cu BlueJ (in Java)**
  - **Tipuri de date, campurile si starea obiectelor**
  - **Comportamentul si interactiunea obiectelor**
  - **Codul sursa Java, editarea si compilarea lui in BlueJ**

- **Studiu de caz: Program de calcul al unui polinom** (variante orientate spre obiecte)

- **Exercitii suplimentare** si **Teme de casa**

2.2. Obiecte si clase (Java)

2.2.1. Definitii

Programele de calcul sunt secvente de instructiuni care prin executia lor pe sisteme (masini) de calcul *rezolva probleme* aparute in diferite domenii *ale lumii reale*.

Un program de calcul scris intr-un limbaj *orientat spre obiecte* reprezinta un **model** al unei parti din lumea reala. *Elementele* care compun modelul (**obiectele software**) sunt construite prin *analogie* cu entitati care apar in lumea reala (obiecte, concepte).

Obiectele software obtinute prin modelare (analogie cu lumea reala) trebuie *reprezentate* in limbajul de programare. Ca in cazul obiectelor si conceptelor din lumea reala, obiectele software pot fi *categorisite*, iar o constructie software (structura) complexa numita **clasa** *descrie* (intr-o forma abstracta) toate obiectele de un tip particular.

La fel ca in lumea reala, in care obiectele si conceptele sunt clasificate pe baza **atributelor** pe care le au acestea, clasele reprezinta obiecte software care au atribute **similare** (atributele fiind elemente de date, variabile interne care caracterizeaza obiectele). De exemplu, la intrarea intr-o sala de curs noi clasificam in mod inconstient obiectele individuale: banci, studenti, ferestre, etc., si interactionam cu ele fara a fi necesar sa le cunoastem toate detaliile (atributele).

Clasa *defineste elementele comune*, **campuri** (atribute) si **metode** (operatii), ale unei categorii de obiecte (reprezinta tipul obiectelor). Toate obiectele clasificate ca banci au latime, inaltime, pozitie in sala, etc. Clasa Banca poate fi definita prin campurile ei ca:

```
class Banca
   latime
   inaltime
   pozitie
```

Obiectul este o *instanta specifica* (un exemplu specific) a unei clase (in care fiecare camp are o anumita valoare), *clasa* fiind *tiparul* dupa care sunt construite obiectele. O sala poate avea 30 de obiecte clasificate ca banci, fiecare banca avand propriile valori ale atributelor latime, inaltime, pozitie in sala, etc. Doua obiecte banca, banca1 si banca2, sunt instante (exemple) diferite ale aceleiasi clase Banca, au in comun aceleasi atribute, dar pot avea diferite valori ale acestor atribute:

banca1

```
latime   80 cm
inaltime 70 cm
pozitie  rand 2, a 3-a
```

banca2

```
latime   120 cm
inaltime 70 cm
pozitie  rand 4, a 6-a
```

In laborator:

1. **Numiti** 2 clase de obiecte din imediata voastra vecinatate in acest moment.
2. **Scrieti numele** fiecarei clase si a cate trei atribute evidente ale fiecarei clase.
3. Pentru cate un obiect din fiecare clasa **definiti valori** ale fiecaruia dintre cele trei campuri.

2.2.2. Crearea obiectelor

Pentru a crea noi instante ale unor banci reale trebuie folosite profile de lemn, metal, etc., dar atunci cand modelam bancile intr-un program de calcul, putem crea doua banci (**in Java**) cu:

Banca, au in comun aceleasi atribute, dar pot avea diferite valori ale acestor atribute:

```
new Banca()
```

```
new Banca()
```

Pentru a putea trata (accesa) distinct cele doua obiecte, este necesara utilizarea a doua nume diferite pentru cele doua obiecte, ceea ce corespunde codului Java:

```
Banca banca1 = new Banca()
```

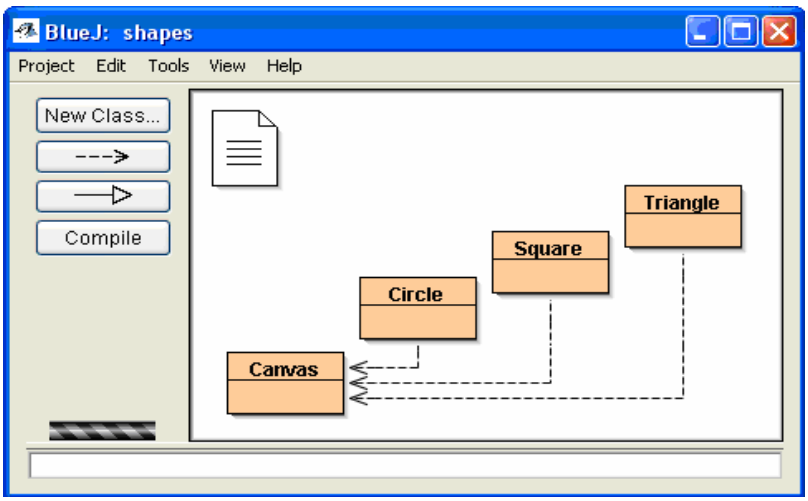
```
Banca banca2 = new Banca()
```

In laborator:

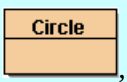
1. **Dati nume pentru 2 obiecte Java** care sa corespunda celor 2 clase anterior numite.
2. **Scrieti in Java** codul pentru crearea celor doua obiecte.

In laborator:

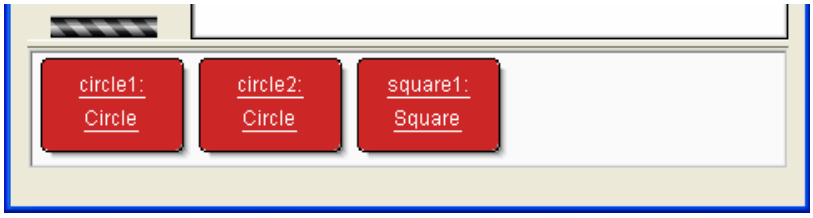
1. **Lansati in executie** mediul de dezvoltare **BlueJ**.
2. **Deschideti proiectul** numit ***shapes***.
   I. Click pe meniul **Project**, apoi selectati **Open Project …** (sau direct **Ctrl+O**)
   II. Selectati succesiv **C:**, **BlueJ**, **examples**, **shapes**, (sau scrieti **C:\BlueJ\examples\shapes**)



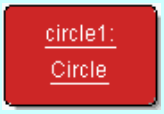
In laborator:

1. **Click-dreapta** (meniul *pop-up*) pe 

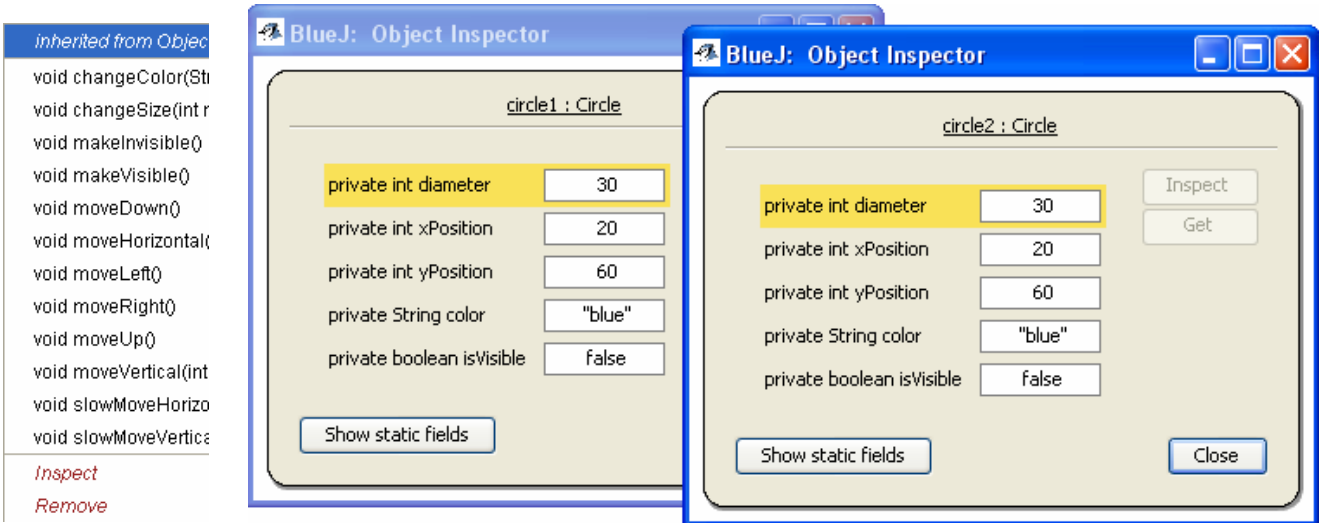
, selectati **new Circle()**, si acceptati valoarea implicita.
2. **Creati un alt cerc**, acceptand din nou valoarea implicita oferita de BlueJ.
3. **Creati un patrat** (Square) in aceleasi conditii.



In laborator:

1. **Click-dreapta** (meniul *pop-up*) pe **primul obiect de tip cerc** 

si selectati ***Inspect***.
2. Repetati operatia pentru **al doilea cerc**. Apoi **comparati valorile atributelor** (campurilor – *fields*).
3. **Creati un patrat** (Square).

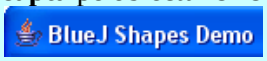


2.2.3. Apelul (invocarea) metodelor

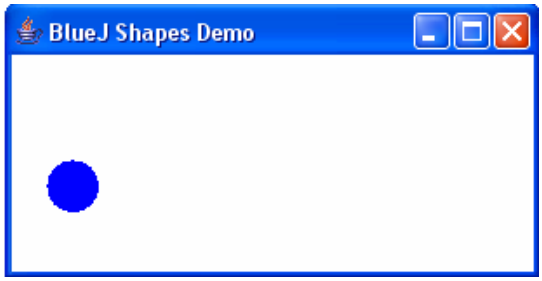
Metoda (operatia) realizeaza o *secventa de actiuni* (reprezentate in programe prin instructiuni) *asupra obiectului* caruia ii apartine, adica asupra *valorilor campurilor obiectului*. **Actiunile** au ca efect *obtinerea valorilor atributelor* obiectului, *modificarea acestor valori*, *realizarea unor sarcini utilizand aceste valorile*, sau *o combinatie* a acestor 3 efecte elementare.

Regruparea elementelor de date (campuri/atribute) si de comportament (metode/operatii) asociate se numeste **incapsulare**. Incapsularea **orientata spre obiecte** permite *ascunderea detaliilor interne* (informatii: campuri/atribute, si implementare: cod metode/operatii) *in spatele unei interfete publice* (setul de semnaturi ale metodelor/operatiilor).

In laborator:

1. **Click-dreapta** pe obiectul **circle1** si selectati **void makeVisible()**.
2. **Click** pe 

pentru urmari **efectul grafic al apelului** metodei **makeVisible()**.
3. **Click-dreapta** pe obiectul **circle1** si selectati **moveUp()**. Urmariti efectul grafic.
4. Repetati de mai multe ori apelul **moveUp()**, urmarind efectul grafic.

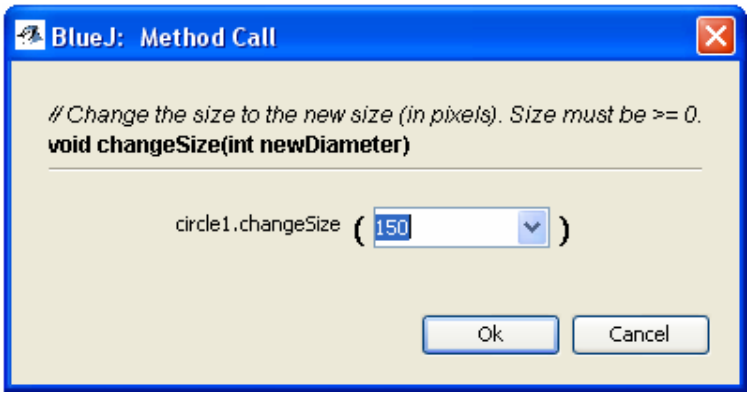


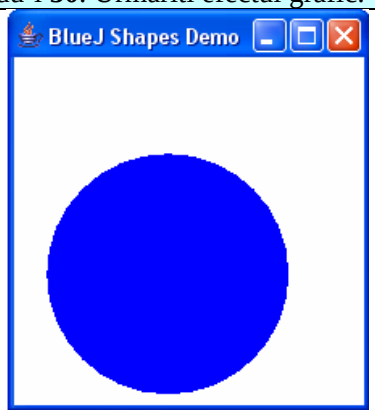
2.2.3. Parametrii metodelor

Parametrii specifica valorile (informatiile) necesare metodelor pentru a fi executate. De exemplu, apelul **circle1.changeSize(50)** specifica valoarea 50 ca parametru (utilizat de metoda **changeSize()** pentru a da valoarea 50 diametrului cercului).

In laborator:

1. **Click-dreapta** pe obiectul **circle1** si selectati **void makeVisible()**.
2. **Click** pe BlueJ Shapes Demo pentru urmari **efectul grafic al apelului** metodei.
3. **Click-dreapta** pe **circle1** si selectati **void changeSize(int newDiameter)**.
4. **Stabiliti valoarea diametrului** la **150** in fereastra care apare pe ecran. Urmariti efectul grafic.
5. **Apelati** metoda **void slowMoveVertical(int distance)** pasandu-i **50**. Urmariti efectul grafic.
6. **Apelati de mai multe ori** metoda **void moveUp()**. Urmariti efectul grafic. Comparati efectele.
7. **Apelati** metoda **void slowMoveHorizontal(int distance)** pasandu-i **50**. Urmariti efectul grafic.

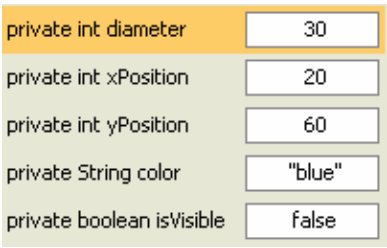




2.2.4. Tipuri de date

Descrierea problemelor reale sub forma de **modele** pentru programme de calcul necesita **definirea datelor problemei**. Urmatoarele campuri descriu obiectul **circle1** de tip **Circle**:

```
circle1
int diameter        30
int xPosition       20
int yPosition       60
String color        "blue"
boolean isVisible   false
```



Tipurile de date specifica **domeniul de definitie al valorilor** campurilor, parametrilor sau variabilelor locale (ale functiilor). Mai exact, tipurile de date specifica :

- **spatiul de memorie alocat** pentru stocarea valorii campului/parametrului/variabilei (de ex., **4B = 32b** pentru tipul **int**, **1b** pentru tipul **boolean**, etc.),

- **gama valorilor** posibile (de ex., $-2^{31}\ldots2^{31}-1$ pentru **int**, valorile **true** si **false** pentru **boolean**),

- **formatul valorilor literale**/de tip imediat (de ex., **100000** sau **-2000** pentru tipul **int**, **true** sau **false** pentru tipul **boolean**, etc.),

- **conventiile privind conversiile** catre alte tipuri (de ex., tipul **int** se poate converti **direct, implicit**, la tipurile **long**, **float** si **double**, si poate fi convertit **explicit, prin *cast*** – conversie prin trunchiere, la tipurile **byte** si **short**, pe cand tipul **boolean** nu poate fi convertit la nici un alt tip, etc.),

- **valorile implicite** (**doar in cazul campurilor!**, **0** pentru tipul **int**, **false** pentru tipul **boolean**),

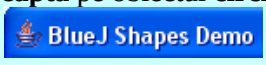
- **operatorii permisi** (pentru astfel de informatii consultati **tutorialul Java oferit de Sun**, in special bazele limbajului Java: http://java.sun.com/docs/books/tutorial/java/nutsandbolts/index.html, **specificatia limbajului Java**: http://java.sun.com/docs/books/jls/second_edition/html/typesValues.doc.html si **mai ales tabelele din introducerea in Java oferita ca anexa la laborator**).

Tipurile de date primitive Java:

	Tip	Valoare implicita	Spatiu memorie	Gama valori	Conversii explicite (cast, trunchiere)	Conversii implicite (extindere)
Valori intregi cu semn	byte	0	8 biti (1B)	-128 … 127	Nu sunt necesare	La short, int, long, float, double
	short	0	16 biti (2B)	-32768 … 32767	La byte	La int, long, float, double
	int	0	32 biti (4B)	-2147483648 … 2147483647	La byte, short	La long, float, double
	long	0l	64 biti (8B)	-9223372036854775808 …9223372036854775807	La byte, short, int	La float, double
Valori in virgula mobile cu semn	float	0.0f	32 biti (4B)	+/-1.4E-45 … +/-3.4028235E+38, +/-infinity, +/-0, NAN	La byte, short, int, long	La double
	double	0.0	64 biti (8B)	+/-4.9E-324 … +/-1.7976931348623157E+308, +/-infinity, +/-0, NaN	La byte, short, int, long, float	Nu exista
Caractere codificate UNICODE	char	\u0000 (null)	16 biti (2B)	\u0000 … \uFFFF	Nu exista	Nu exista
Valori logice	boolean	`false`	1 bit folosit din 32 biti	true, false	Nu exista	Nu exista

Pentru exemple de **tipuri de date complexe**, numite tipuri **referinta** (deoarece accesul la variabilele de acel tip se face prin referinte), **clase Java**, printre care si clasa **String**, consultati **anexa la laborator dedicata claselor de bibioteca Java mai reprezentative**.

In laborator:

1. **Click-dreapta** pe obiectul **circle1** si selectati **void makeVisible()**.
2. **Click** pe 

pentru urmari **efectul grafic al apelului** metodei.
3. **Apelati** metoda **void changeColor(String newColor)** pasandu-i **`"red"`**. Urmariti efectul grafic.
4. **Apelati** metoda **void changeColor(String newColor)** pasandu-i **`"rosu"`**. **Ce observati?**
5. **Apelati** metoda **void changeColor(String newColor)** pasandu-i **`red`**. **Ce observati?**

2.2.5. Instante (obiecte) multiple

Folosind definitia unei clase (de ex. Circle) **pot fi create mai multe obiecte** de acelasi tip (diferentiate/identificate prin nume):

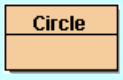
`circle1`

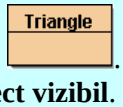
```
int diameter        30
int xPosition       20
int yPosition       60
String color        "blue"
boolean isVisible   false
```

`circle2`

```
int diameter        30
int xPosition       20
int yPosition       60
String color        "blue"
boolean isVisible   false
```

In laborator:

1. **Creati trei obiecte Circle cu click-dreapta** pe 

si **new Circle()**.
2. **Faceti fiecare obiect vizibil**. **Deplasati** obiectele. **Schimbati culorile** obiectelor.
3. **Creati trei obiecte** 

.
4. **Faceti fiecare obiect vizibil**. **Deplasati** obiectele. **Schimbati culorile** obiectelor.

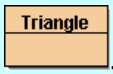
2.2.6. Starea unui obiect

Ansamblul valorilor campurilor (atributelor) **unui obiect la un moment dat** reprezinta **starea obiectului** (poate diferi in timp, ca urmare a comportamentului, poate fi diferita la un moment dat pentru diferite obiecte de acelasi tip):

```
circle1
int diameter        30
int xPosition       20
int yPosition       60
String color        "blue"
boolean isVisible   false
```

In laborator:

1. **Inspectati starea** obiectului **circle1** cu **double-click** pe 

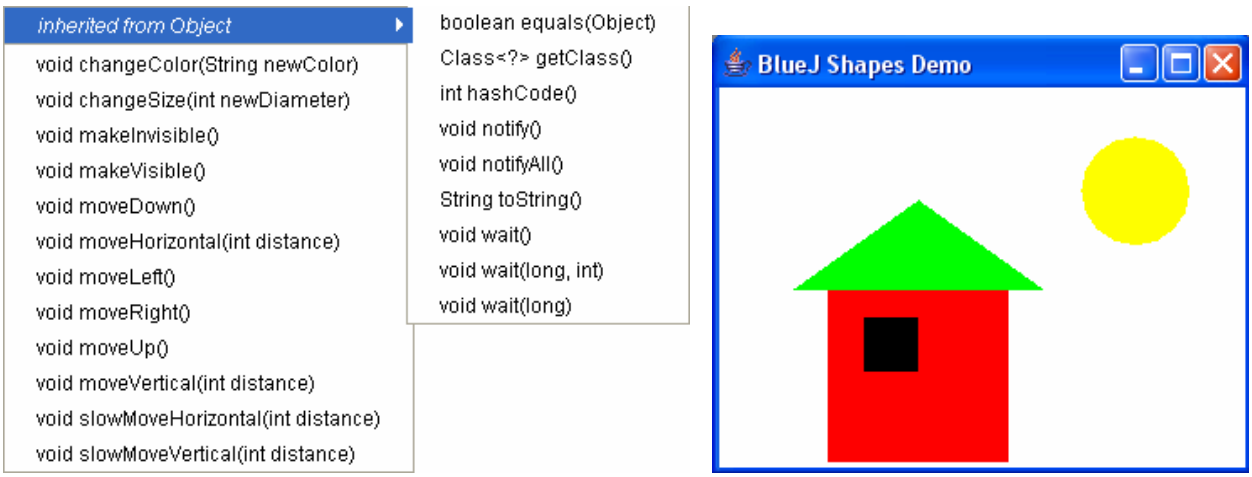
(sau **click-dreapta** si ***Inspect***).
2. **Schimbati culoarea** obiectului **circle1**si **inspectati-i din nou starea (valorile campurilor)**.
3. **Creati doua obiecte** 

. Inspectati-le starea.
4. **Au toate campurile aceleasi nume? Sunt toate valorile aceleasi?**
5. **Apelati metode care schimba pozitia** celor doua obiecte. Inspectati-le starea. **Ce s-a schimbat?**
6. **Creati doua obiecte din clase diferite.** Inspectati-le starea. **Ce campuri au aceleasi nume?**

2.2.7. Comportamentul unui obiect

O **metoda** realizeaza **o actiune** (complexa, secventa de actiuni elementare) **asupra** (valorilor campurilor) **obiectului, utilizand valorile campurilor obiectului**, astfel **efectuand o sarcina pentru codul care a apelat** la acea metoda (cod **context**). Ea este un **atom de comportament** al obiectului, **comportamentul global** al obiectului fiind obtinut prin **inlantuirea apelurilor de metode**.

Toate **obiectele din aceeasi clasa** au **aceleasi metode disponibile**. De exemplu, clasa **Circle** are urmatoare metode:



In laborator:

1. **Creati o imagine care sa schiteze o casa si un soare similare celor din imaginea de mai sus**.
2. **Notati-va sarcinile pe care le-ati indeplinit pentru a obtine acest efect**. De exemplu:

```
I.   Circle circle1 = new Circle()
II.  circle1 makeVisible()
III. circle1 moveHorizontal(200)
IV.  circle1 changeSize(50)
V.   circle1 changeColor("yellow")
VI.  ...
```

3. **Ar fi putut fi apelate metodele in alta ordine pentru a obtine acelasi efect?**

2.2.6. Interactiunea obiectelor

Sarcinile realizate manual in exercitiul anterior **sunt in mod normal scrise sub forma de instructiuni Java** intr-un fisier, **pentru a putea fi executate din nou**. Primii 5 pasi ar fi scrisi in Java:

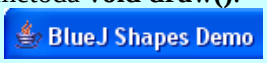
```
Circle circle1 = new Circle();
circle1.makeVisible();
circle1.moveHorizontal(200);
circle1.changeSize(50);
circle1.changeColor("yellow");
```

BlueJ ofera un exemplu de program (proiectul ***picture***) care contine pe langa clasele **Canvas**, **Circle**, **Square** si **Triangle** si codul unei clase **Picture** realizeaza desenul de mai sus prin crearea obiectelor necesare si apelul metodelor respectivelor obiecte, astfel incat sa fie pozitionate, dimensionate si colorate ca in desen.

Se poate spune ca **obiectul de tip Picture** interactioneaza (**colaboreaza**, comunica **prin mesaje** – **apelurile de metode**) **cu obiectele** de tip **Circle**, **Square** si **Triangle** pentru a realiza sarcina globala.

In laborator:

1. **Deschideti proiectul** numit ***pictures*** (**Ctrl-O**, apoi pe **C:\BlueJ\examples** selectati **picture**)
2. **Creati un obiect** 

.
3. **Apelati** metoda **void draw()**.
4. **Click** pe 

pentru urmari **efectul grafic al apelului** metodei.

2.2.7. Codul sursa Java. Editarea si compilareacu BlueJ

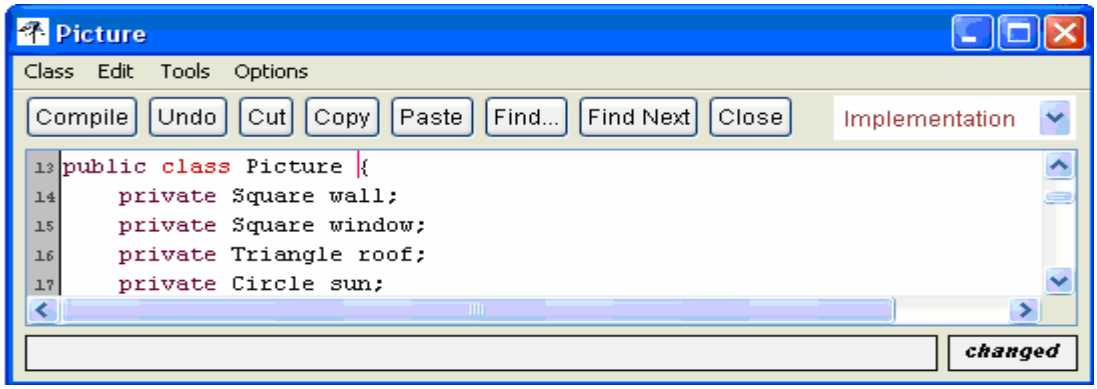
Sarcinile pentru crearea obiectelor si apelul metodelor pot fi **scrise sub forma de instructiuni Java**, **salvate** intr-un fisier, **si reutilizate** cand este nevoie de ele.

Listele instructiunilor Java (grupate **in metode**, iar acestea **impreuna cu campurile**) **definesc o clasa Java**. Textul scris al instructiunilor formeaza **codul sursa al clasei**.

In laborator:

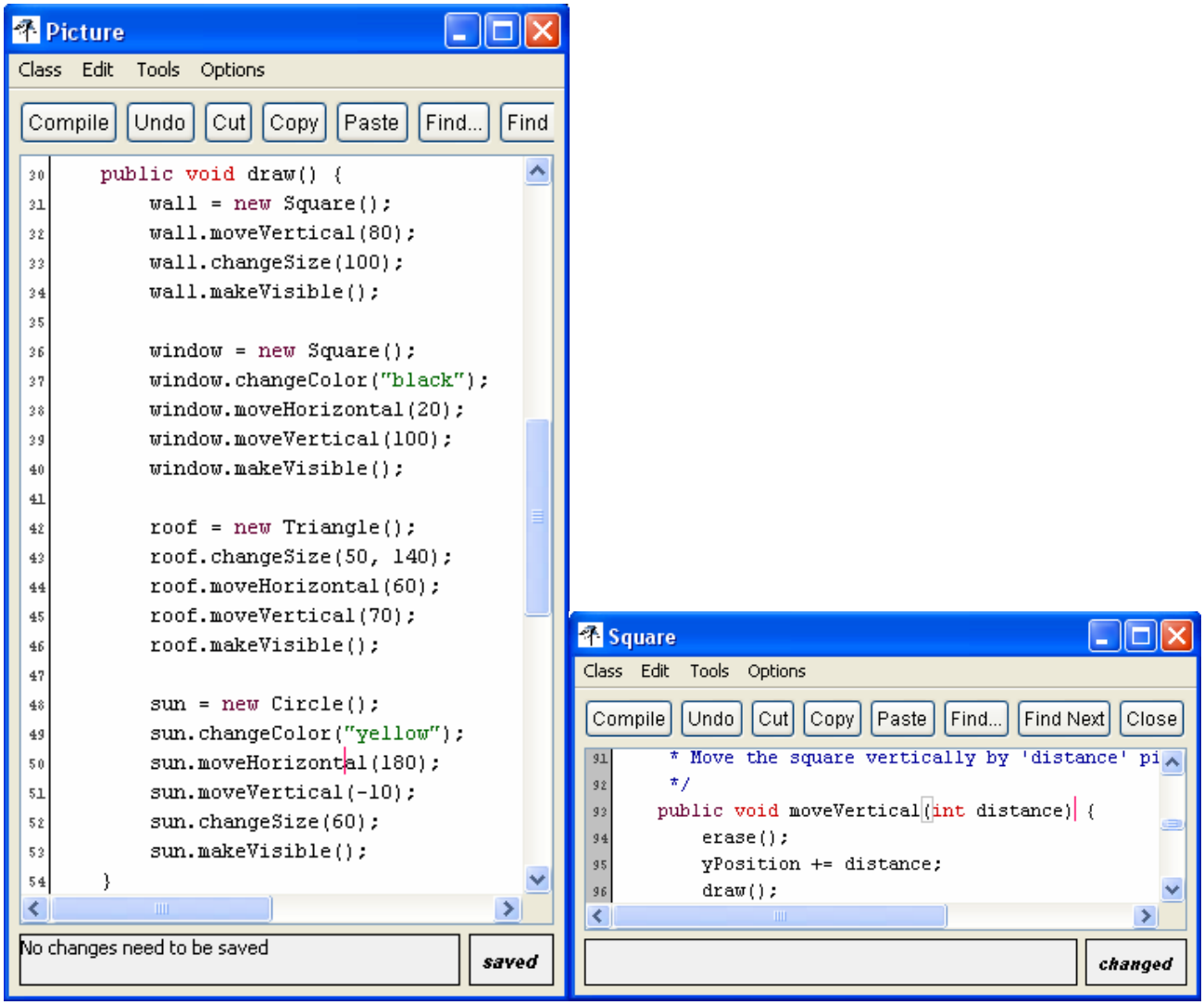
1. **Deschideti proiectul** numit ***pictures***.
2. **Vizualizati codul sursa** al clasei **Picture**, fie **double-click pe** 

, fie **right-click**, **Open Editor**.
3. **Care este numele clasei?** Gasiti **instructiunea care defineste numele** clasei.
4. **Gasiti campurile** pentru **soare** si **partile componente ale casei**. Observati **cum sunt declarate.**



In laborator:

5. **Gasiti codul metodei** a carei declaratie (**semnatura**) este **public void draw()**.
6. **Care sunt sarcinile elementare** (instructiunile de tip apel) **indeplinite pentru a crea zidul?**
7. **Conteaza ordinea in care sunt efectuate** aceste sarcini?
8. **Vizualizati codul sursa** al clasei **Square**. **Gasiti semnaturile metodelor invocate in** metoda **draw()** a clasei **Picture**. Care sunt **sarcinile indeplinite de codurile acestor metode?**



Instructiunile Java pot fi scrise si citite de programatori, dar **pentru a fi executate trebuie compilate** (translatate) **la cod de octeti Java** (executat apoi de interpretorul **java**).

In laborator:

1. **Vizualizati codul sursa** al clasei **Picture**. Gasiti codul sursa al metodei **public void draw()**.
2. **Modificati culoarea zidului** in **`"blue"`**.
3. **Compilati codul sursa** cu **click pe** butonul **Compile**.
4. **Ce s-a intamplat cu** obiectul **picture1**?
5. **Creati un obiect** al clasei **Picture**. **Click** pe BlueJ Shapes Demo si urmariti **efectul**.

2.3. Varianta orientata spre obiecte a programului de calcul al unui polinom

2.3.1. Program de calcul al unui polinom (de la delegarea interna catre metode la prima varianta cu delegare catre un alt obiect)

Programul cerut ca tema la lucrarea trecuta, numita **`Polinom2`**, **utilizeaza delegarea unor sarcini** (*tasks*) **catre metode publice, declarate global la nivelul clasei** (declarate `public static`):

```
 1    import javax.swing.JOptionPane;
 2    public class Polinom2 {
 3
 4      // Metoda care obtine de la utilizator gradul polinomului
 5      public static int obtineGrad() {
 6        // Obtinerea de la utilizator a gradului polinomului
 7        int gradPolinom = Integer.parseInt(JOptionPane.showInputDialog(
 8                                           "Introduceti gradul polinomului"));
 9        // Returnarea valorii gradului polinomului
10        return (gradPolinom);
11      }
12
13      // Metoda care obtine de la utilizator coeficientii polinomului
14      public static int[] stabilesteCoeficienti(int gradPolinom) {
15        // Declararea si crearea tabloului coeficientilor, numit coeficienti
16        int[] coeficienti = new int[gradPolinom+1];
17
18        // Obtinerea de la utilizator a coeficientilor Ci, unde i=0,N
19        for (int i=0; i<=gradPolinom; i++) {
20          coeficienti[i] = Integer.parseInt(JOptionPane.showInputDialog(
21                                              "Coeficientul de grad " + i));
22        }
23        // Returnarea tabloului coeficientilor
24        return (coeficienti);
25      }
26
27      // Metoda care afiseaza polinomul P(X)
28      public static void afisarePolinom(int gradPolinom, int[] coeficienti) {
29        // Afisarea polinomului P(X)
30        // - mai intai termenul liber Co
31        System.out.print("P(X) = " + coeficienti[0]);
32
33        // - apoi termenii Ci*X^i, unde i=1,N
34        for (int i=1; i<=gradPolinom; i++) {
35          System.out.print(" + " + coeficienti[i]  + "*X^" + i);
36        }
37        System.out.println();
38      }
39
40      // Metoda care obtine de la utilizator valoarea necunoscutei
41      public static int obtineNecunoscuta() {
42        // Obtinerea de la utilizator a valorii necunoscutei
43        int necunoscuta = Integer.parseInt(JOptionPane.showInputDialog(
44                                             "Valoarea necunoscutei"));
45        // Returnarea valorii necunoscutei
46        return (necunoscuta);
47      }
48
49      // Metoda care calculeaza valoarea polinomului pt o valoare a necunoscutei
50      public static int valoarePolinom(int gradPolinom, int[] coeficienti,
51                                                        int necunoscuta) {
52        // Declararea si initializarea variabilei intregi numita polinom,
```

```
53        // care contine valoarea polinomului, P(X)
54        int polinom = 0;
55        int X_i = 1;
56
57        // Calculul polinomului P(X) = suma(Ci * X^i), unde i=0,N
58        for (int i=0; i<=gradPolinom; i++) {
59           // - actualizarea valorii polinomului
60           polinom = polinom + coeficienti[i]*X_i;
61           // - actualizarea valorii X^i, unde i=1,N
62           X_i = X_i * necunoscuta;
63        }
64        // Returnarea valorii polinomului
65        return (polinom);
66      }
67
68      // Metoda principala. Utilizata pentru testarea celorlalte metode.
69      public static void main(String[] args) {
70        // Apelul metodei care obtine de la utilizator gradul polinomului
71        int grad = obtineGrad();
72
73        // Apelul metodei care obtine de la utilizator coeficientii polinomului
74        int[] coeficienti = stabilesteCoeficienti(grad);
75
76        // Apelul metodei care afiseaza polinomul
77        afisarePolinom(grad, coeficienti);
78
79        // Apelul metodei care obtine o valoare a necunoscutei
80        int necunoscuta = obtineNecunoscuta();
81
82        // Afisarea valorii necunoscutei
83        System.out.println("X = " + necunoscuta);
84
85        // Apelul metodei care calculeaza polinomul pentru necunoscuta data
86        int polinom = valoarePolinom(grad, coeficienti, necunoscuta);
87
88        // Afisarea valorii polinomului
89        System.out.println("P(" + necunoscuta + ") = " + polinom);
90
91        System.exit(0); // Inchiderea interfetei grafice
92      }
93    }
```

Metodele clasei **`Polinom2`** fiind toate declarate **`static`** pot fi apelate direct de metoda principala (si ea declarata **`static`**).

In laborator:

1. **Compilati si executati programul de mai sus. In BlueJ:**
   - I. Inchideti proiectele anterioare (cu **Project** si **Close** sau **Ctrl+W**).
   - II. **Creati un nou proiect** numit **Polinom2** (cu **Project**, apoi **New Project…**, selectati **D:/**, apoi **Software2006**, apoi **numarul grupei**, apoi scrieti **Polinom2**).
   - III. **Creati o noua clasa**, numita **Polinom2**, apasand **New Class…**
   - IV. **Double-click pe noua clasa** (ii **deschideti codul in editor**), si inlocuiti codul cu cel de sus.
   - V. **Compilati codul** si **executati** metoda **public static void main(String[] args)**.
2. **Stergeti cuvintele `static` din declaratiile metodelor (cu exceptia metodei principale, main).**
3. **Recompilati codul. Ce observati? Care este cauza probabila?**

Practic, **in clasa `Polinom2` nu sunt create obiecte noi, iar metodele apelate tin de clasa sin u de obiecte, asa incat nu se poate vorbi despre orientare spre obiecte** pura.

Pentru a se lucra cu obiecte, ar trebui folosite metode non-statice, iar **pentru a avea interactiune intre obiecte** o **alta clasa ar trebui sa creeze** obiecte **Polinom** si **sa apeleze metodele** acestui obiect.

Pornind de la programul de mai sus, **se poate scrie codul unei clase Java** numita **`Polinom3`**, a carei **structura interna contine**:

- **o metoda** (declarata `public`) numita **`obtineGrad()`**, care obtine de la utilizator valoarea gradului polinomului, si o **returneaza** ca intreg de tip **`int`**,

- **o metoda** (declarata `public`) numita **`stabilesteCoeficienti()`**, care **primeste** un parametru intreg de tip **`int`**, numit **`gradPolinom`**, reprezentand gradului polinomului, creaza un nou tablou al coeficientilor (cu **`gradPolinom +1`** elemente), obtine de la utilizator valori pentru coeficientii polinomului si populeaza cu ei tabloul nou creat, apoi **returneaza** taboul de tip **`int[]`** creat,

- **o metoda** (declarata `public`) numita **`obtineNecunoscuta()`**, care obtine de la utilizator valoarea necunoscutei, si o **returneaza** ca intreg de tip **`int`**,

- **o metoda** (declarata `public`) numita **`afisarePolinom()`**, care primeste un parametru intreg de tip **`int`**, numit **`gradPolinom`**, reprezentand gradului polinomului, si un parametru de tip **`int[]`**, numit **`coeficienti`**, reprezentand coeficientii polinomului, si **afiseaza** polinomul corespunzator valorilor primite,

- **o metoda** (declarata `public`) numita **`valoarePolinom()`**, care primeste un parametru intreg de tip **`int`**, numit **`gradPolinom`**, reprezentand gradului polinomului, un parametru de tip **`int[]`**, numit **`coeficienti`**, reprezentand coeficientii polinomului, si un parametru intreg de tip **`int`**, numit **`necunoscuta`**, reprezentand necunoscuta, apoi calculeaza valoarea polinomului corespunzatoare valorilor primite si o **returneaza** ca intreg de tip **`int`**.

```
 1    import javax.swing.JOptionPane;
 2    public class Polinom3 {
 3
 4      // Metoda care obtine de la utilizator gradul polinomului
 5      public int obtineGrad() {
 6        // Obtinerea de la utilizator a gradului polinomului
 7        int gradPolinom = Integer.parseInt(JOptionPane.showInputDialog(
 8                                           "Introduceti gradul polinomului"));
 9        // Returnarea valorii gradului polinomului
10        return (gradPolinom);
11      }
12
13      // Metoda care obtine de la utilizator coeficientii polinomului
14      public int[] stabilesteCoeficienti(int gradPolinom) {
15        // Declararea si crearea tabloului coeficientilor, numit coeficienti
16        int[] coeficienti = new int[gradPolinom+1];
17
18        // Obtinerea de la utilizator a coeficientilor Ci, unde i=0,N
19        for (int i=0; i<=gradPolinom; i++) {
20          coeficienti[i] = Integer.parseInt(JOptionPane.showInputDialog(
21                                                "Coeficientul de grad " + i));
22        }
23        // Returnarea tabloului coeficientilor
24        return (coeficienti);
25      }
26
27      // Metoda care afiseaza polinomul P(X)
28      public void afisarePolinom(int gradPolinom, int[] coeficienti) {
29        // Afisarea polinomului P(X)
30        // - mai intai termenul liber Co
31        System.out.print("P(X) = " + coeficienti[0]);
32
33        // - apoi termenii Ci*X^i, unde i=1,N
34        for (int i=1; i<=gradPolinom; i++) {
35          System.out.print(" + " + coeficienti[i]  + "*X^" + i);
36        }
37        System.out.println();
38      }
39
```

```
40      // Metoda care obtine de la utilizator valoarea necunoscutei
41      public int obtineNecunoscuta() {
42        // Obtinerea de la utilizator a valorii necunoscutei
43        int necunoscuta = Integer.parseInt(JOptionPane.showInputDialog(
44                                                    "Valoarea necunoscutei"));
45        // Returnarea valorii necunoscutei
46        return (necunoscuta);
47      }
48
49      // Metoda care calculeaza valoarea polinomului pt o valoare a necunoscutei
50      public int valoarePolinom(int gradPolinom, int[] coeficienti,
51                                                      int necunoscuta) {
52        // Declararea si initializarea variabilei intregi numita polinom,
53        // care contine valoarea polinomului, P(X)
54        int polinom = 0;
55        int X_i = 1;
56
57        // Calculul polinomului P(X) = suma(Ci * X^i), unde i=0,N
58        for (int i=0; i<=gradPolinom; i++) {
59           // - actualizarea valorii polinomului
60           polinom = polinom + coeficienti[i]*X_i;
61           // - actualizarea valorii X^i, unde i=1,N
62           X_i = X_i * necunoscuta;
63        }
64        // Returnarea valorii polinomului
65        return (polinom);
66      }
67    }
```

Metodele de mai sus (ale **obiectelor din clasa Polinom3**) **sunt obtinute din cele ale clasei Polinom2 prin simpla indepartare a cuvintelor cheie static din semnaturile metodelor.**

Se poate scrie apoi codul unei clase Java numita **`RunPolinom3`**, care **contine** o **metoda principala, de test**, care specifica **scenariul principal**:

- **creeaza un obiect** numit **`poli3`** de tip **`Polinom3`** (al clasei curente), **folosind constructorul fara parametri (**numit **`Polinom3()`**), **oferit implicit de masina virtuala Java (JVM)**,

- **deleaga** obtinerea valorii gradului polinomului catre metoda **`obtineGrad()`** a obiectului **`poli3`**,

- **deleaga** stabilirea valorilor coeficientilor polinomului catre metoda **`stabilesteCoeficienti()`** a obiectului **`poli3`**,

- **deleaga** afisarea polinomului catre metoda **`afisarePolinom()`** a obiectului **`poli3`**,

- **deleaga** obtinerea valorii necunoscutei catre metoda **`obtineNecunoscuta()`** a obiectului **`poli3`**,

- afiseaza valoarea necunoscutei,

- **deleaga** calculul valorii polinomului catre metoda **`valoarePolinom()`** a obiectului **`poli3`**,

- afiseaza valoarea polinomului.

```
1   import javax.swing.JOptionPane;
2   public class RunPolinom3 {
3
4     private static Polinom3 poli3;
5
6     // Metoda principala. Utilizata pentru testarea clasei Polinom3.
7     public static void main(String[] args) {
8
9       // Crearea unui obiect al clasei Polinom3
10      poli3 = new Polinom3();
11
12      // Apelul metodei care obtine de la utilizator gradul polinomului
13      int grad = poli3.obtineGrad();
14
15      // Apelul metodei care obtine de la utilizator coeficientii polinomului
16      int[] coeficienti = poli3.stabilesteCoeficienti(grad);
```

```
17
18        // Apelul metodei care afiseaza polinomul
19        poli3.afisarePolinom(grad, coeficienti);
20
21        // Apelul metodei care obtine o valoare a necunoscutei
22        int necunoscuta = poli3.obtineNecunoscuta();
23
24        // Afisarea valorii necunoscutei
25        System.out.println("X = " + necunoscuta);
26
27        // Apelul metodei care calculeaza polinomul pentru necunoscuta data
28        int polinom = poli3.valoarePolinom(grad, coeficienti, necunoscuta);
29
30        // Afisarea valorii polinomului
31        System.out.println("P(" + necunoscuta + ") = " + polinom);
32
33        System.exit(0); // Inchiderea interfetei grafice
34      }
35    }
```

In laborator:

1. **Compilati si executati programul de mai sus** in BlueJ (**creati un proiect nou** cu numele **Polinom3**, si **doua clase** numite **Polinom3** si **RunPolinom3 in care copiati codul de mai sus).**

Se observa ca **in codul metodei principale**, ca si in cazul clasei Polinom2, **este necesara stocarea sub forma unor variabile locale a valorilor `gradPolinom`, `coeficienti`, `necunoscuta` si `polinom`**.

Structura unui polinom depinde insa **doar de valorile `gradPolinom` si `coeficienti`**, si de aceea **este recomandabila plasarea lor ca atribute (campuri) alaturi de metodele care fac initializarea, afisarea si calculul polinomului**.

Celelalte doua valori `necunoscuta` si `polinom` pot ramane exterioare clasei Polinom, valoarea **`necunoscuta`** fiind **independenta de structura polinomului** iar valoarea **`polinom`** fiind **calculata pe baza elementelor de structura ale polinomului** (`gradPolinom` si `coeficienti`).

2.3.2. Program de calcul al unui polinom (a doua varianta cu delegare catre un obiect)

Se poate scrie codul unei clase Java numita **`Polinom4`**, a carei **structura interna contine**:

- **un camp** (declarat `private`) intreg de tip **`int`** numit **`gradPolinom`**, care contine gradului polinomului,
- **un camp** (declarat `private`) intreg de tip **`int[]`** numit **`coeficienti`**, care contine coeficientii polinomului,
- **o metoda** (declarata `public`) numita **`obtineGrad()`**, care obtine de la utilizator valoarea gradului polinomului, si o foloseste pentru a **da valoare campului `gradPolinom`**,
- **o metoda** (declarata `public`) numita **`stabilesteCoeficienti()`**, care foloseste valoarea campului **`gradPolinom`**, pentru a crea un nou tablou cu **`(gradPolinom+1)`** elemente, pe care **il atribuie campului `coeficienti`**, apoi obtine de la utilizator valori pentru coeficientii polinomului si populeaza cu ei tabloul nou creat,
- **o metoda** (declarata `public`) numita **`obtineNecunoscuta()`**, care obtine de la utilizator valoarea necunoscutei, si o **returneaza** ca intreg de tip **`int`**,
- **o metoda** (declarata `public`) numita **`afisarePolinom()`**, care foloseste campurile **`gradPolinom`** si **`coeficienti`** pentru a **afisa** polinomul corespunzator valorilor respective,
- **o metoda** (declarata `public`) numita **`valoarePolinom()`**, care primeste un parametru intreg de tip **`int`**, numit **`necunoscuta`**, reprezentand necunoscuta, si foloseste campurile **`gradPolinom`** si **`coeficienti`** pentru a calcula valoarea polinomului pentru valoarea primita a necunoscutei si a o **returna** ca intreg de tip **`int`**,

```
 1  import javax.swing.JOptionPane;
 2  public class Polinom4 {
 3
 4    // Campuri (atribute, variabile membru)
 5    int gradPolinom;
 6    int[] coeficienti;
 7
 8    // Metoda care obtine de la utilizator gradul polinomului
 9    public void obtineGrad() {
10      // Obtinerea de la utilizator a gradului polinomului
11      gradPolinom = Integer.parseInt(JOptionPane.showInputDialog(
12                                          "Introduceti gradul polinomului"));
13    }
14
15    // Metoda care obtine de la utilizator coeficientii polinomului
16    public void stabilesteCoeficienti() {
17      // Declararea si crearea tabloului coeficientilor, numit coeficienti
18      coeficienti = new int[gradPolinom+1];
19
20      // Obtinerea de la utilizator a coeficientilor Ci, unde i=0,N
21      for (int i=0; i<=gradPolinom; i++) {
22        coeficienti[i] = Integer.parseInt(JOptionPane.showInputDialog(
23                                              "Coeficientul de grad " + i));
24      }
25    }
26
27    // Metoda care afiseaza polinomul P(X)
28    public void afisarePolinom() {
29      // Afisarea polinomului P(X)
30      // - mai intai termenul liber Co
31      System.out.print("P(X) = " + coeficienti[0]);
32
33      // - apoi termenii Ci*X^i, unde i=1,N
34      for (int i=1; i<=gradPolinom; i++) {
35        System.out.print(" + " + coeficienti[i]  + "*X^" + i);
36      }
37      System.out.println();
38    }
39
40    // Metoda care obtine de la utilizator valoarea necunoscutei
41    public int obtineNecunoscuta() {
42      // Obtinerea de la utilizator a valorii necunoscutei
43      int necunoscuta = Integer.parseInt(JOptionPane.showInputDialog(
44                                                "Valoarea necunoscutei"));
45      // Returnarea valorii necunoscutei
46      return (necunoscuta);
47    }
48
49    // Metoda care calculeaza valoarea polinomului pt o valoare a necunoscutei
50    public int valoarePolinom(int necunoscuta) {
51      // Declararea si initializarea variabilei intregi numita polinom,
52      // care contine valoarea polinomului, P(X)
53      int polinom = 0;
54      int X_i = 1;
55      // Calculul polinomului P(X) = suma(Ci * X^i), unde i=0,N
56      for (int i=0; i<=gradPolinom; i++) {
57         // - actualizarea valorii polinomului
58         polinom = polinom + coeficienti[i]*X_i;
59         // - actualizarea valorii X^i, unde i=1,N
60         X_i = X_i * necunoscuta;
61      }
62      // Returnarea valorii polinomului
63      return (polinom);
64    }
65  }
```

Se observa aparitia campurilor (atributelor) **`gradPolinom` si `coeficienti`**.

Se poate scrie apoi codul unei clase Java numita **`RunPolinom4`**, care **contine**:

- o **metoda principala, de test**, care specifica **scenariul**:

- **creeaza un obiect** numit **`poli4`** de tip **`Polinom4`** (al clasei curente), **folosind constructorul fara parametri (**numit **`Polinom4()`), oferit implicit de masina virtuala Java (JVM)**,

- **deleaga** obtinerea valorii gradului polinomului catre metoda **`obtineGrad()`** a obiectului **`poli3`**,

- **deleaga** stabilirea valorilor coeficientilor polinomului catre metoda **`stabilesteCoeficienti()`** a obiectului **`poli3`**,

- **deleaga** afisarea polinomului catre metoda **`afisarePolinom()`** a obiectului **`poli3`**,

- **deleaga** obtinerea valorii necunoscutei catre metoda **`obtineNecunoscuta()`** a obiectului **`poli3`**,

- afiseaza valoarea necunoscutei,

- **deleaga** calculul valorii polinomului catre metoda **`valoarePolinom()`** a obiectului **`poli3`**,

- afiseaza valoarea polinomului.

```
 1   import javax.swing.JOptionPane;
 2   public class RunPolinom4 {
 3
 4     private static Polinom4 poli4;
 5
 6     // Metoda principala. Utilizata pentru testarea clasei Polinom4.
 7     public static void main(String[] args) {
 8
 9       // Crearea unui obiect al clasei Polinom4
10       poli4 = new Polinom4();
11
12       // Apelul metodei care obtine de la utilizator gradul polinomului
13       poli4.obtineGrad();
14
15       // Apelul metodei care obtine de la utilizator coeficientii polinomului
16       poli4.stabilesteCoeficienti();
17
18       // Apelul metodei care afiseaza polinomul
19       poli4.afisarePolinom();
20
21       // Apelul metodei care obtine o valoare a necunoscutei
22       int necunoscuta = poli4.obtineNecunoscuta();
23
24       // Afisarea valorii necunoscutei
25       System.out.println("X = " + necunoscuta);
26
27       // Apelul metodei care calculeaza polinomul pentru necunoscuta data
28       int polinom = poli4.valoarePolinom(necunoscuta);
29
30       // Afisarea valorii polinomului
31       System.out.println("P(" + necunoscuta + ") = " + polinom);
32
33       System.exit(0); // Inchiderea interfetei grafice
34     }
35   }
```

Se observa disparitia variabilelor locale `gradPolinom` si `coeficienti`, in acest fel **metoda principala fiind eliberata de sarcina gestiunii** acestora.

In laborator:

1. **Compilati si executati programul de mai sus** in BlueJ (**creati un proiect nou** cu numele **Polinom4**, si **doua clase** numite **Polinom4** si **RunPolinom4 in care copiati codul de mai sus).**

2.4. Exercitii suplimentare

In introducerea in obiecte si clase Java (dupa R. Baldwin) se dau codurile mai multor clase Java:
- **Radio001** pentru versiunea in care **tot codul este scris in** metoda **main()**,
- **Radio002** pentru versiunea **orientata spre proceduri**,
- **Radio003** pentru versiunea "**orientata spre clase**", si
- **Radio si Radio01** pentru versiunea **orientata spre obiecte**.

Pentru fiecare dintre aceste versiuni:
Compilati si executati programele in BlueJ (**creati proiecte noi** cu numele **Radio001, Radio002, Radio003, si Radio01,** apoi **creati clase cu numele de mai sus**, si **copiati in ele codurile din anexa**).

2.5. Teme pentru acasa

Fiecare student va aduce pentru data viitoare codurile Java pentru 2 clase, aceste coduri continand cel putin 3 campuri/atribute si 3 metode/operatii.

Clasele propuse sunt:

Nume clasa	Exemplu de declaratie de camp (atribut)	Exemplu de declaratie (semnatura) de metoda (operatie)
Scrisoare	`String destinatar`	`void setDestinatar(String nume)`
Mail	`String subject`	`void setSubject(String text)`
Masina	`String proprietar`	`void setProprietar (String nume)`
Bicicleta	`double vitezaCurenta`	`void setVitezaCurenta(double viteza)`
Avion	`int numarMotoareActive`	`void defectiuneMotor()`
Caiet	`int numarFoi`	`void rupereFoaie()`
Clipboard	`int numarFoi`	`void adaugareFoi(int foiNoi)`
SalaCurs	`int locuriOcupate`	`void asezareStudent(String nume)`
Laborator	`int numarPlatforme`	`void adaugarePlatforma()`

Fiecare student va primi un numar de ordine, iar fiecarui numar de ordine ii corespunde un set de 2 clase, conform tabelului urmator:

Nr. ord.	Setul de clase (tema de casa)	Nr. ord.	Setul de clase (tema de casa)	Nr. ord.	Setul de clase (tema de casa)
1	Scrisoare + Masina	**11**	Mail + Caiet	**21**	Bicicleta + SalaCurs
2	Scrisoare + Bicicleta	**12**	Mail + Clipboard	**22**	Bicicleta + Laborator
3	Scrisoare + Avion	**13**	Mail + SalaCurs	**23**	Avion + Caiet
4	Scrisoare + Caiet	**14**	Mail + Laborator	**24**	Avion + Clipboard
5	Scrisoare + Clipboard	**15**	Masina + Caiet	**25**	Avion + SalaCurs
6	Scrisoare + SalaCurs	**16**	Masina + Clipboard	**26**	Avion + Laborator
7	Scrisoare + Laborator	**17**	Masina + SalaCurs	**27**	Caiet + SalaCurs
8	Mail + Masina	**18**	Masina + Laborator	**28**	Caiet + Laborator
9	Mail + Bicicleta	**19**	Bicicleta + Caiet	**29**	Clipboard + SalaCurs
10	Mail + Avion	**20**	Bicicleta + Clipboard	**30**	Clipboard + Laborator