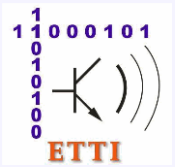
SwRTc ISw
POO – an II

Catedra de Telecomunicatii



25/10/2010

Limbaje de Programare pentru Aplicatii Internet (LPAI)

Laborator 2

Introducere in programarea orientata spre obiecte (OO) in Java

2.1. Descrierea laboratorului

In aceasta lucrare de laborator vor fi acoperite urmatoarele probleme:

- **Utilizarea mediului de dezvoltare integrat** (IDE) **BlueJ** (vezi si ***Tutorial BlueJ in limba romana***)
  - Crearea obiectelor si invocarea metodelor Java cu BlueJ
  - Tipuri de date Java si starea unui obiect
  - Comportamentul unui obiect si interactiunea obiectelor
  - Codul sursa Java, editarea si compilarea lui in BlueJ
- **Studiu de caz: Clasa care incapsuleaza informatii despre un student**
- **Teme de casa** si anexa **Instalarea kitului BlueJ**

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. Programele sunt **solutii** ale acestor probleme. **Un program** scris intr-un limbaj **orientat spre obiecte** (**OO**) reprezinta un **model** al unei parti din lumea reala.

Elementele care compun modelul (numite **OBIECTE** software) sunt construite prin analogie cu entitati care apar in lumea reala (obiecte reale, concepte). Obiectele software obtinute prin modelare (analogie cu lumea reala) trebuie reprezentate in limbajul de programare.

Ca si in cazul obiectelor si conceptelor din lumea reala, obiectele software pot fi **categorisite.** 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 esentiale** pe care le au acestea, **clasele reprezinta obiecte software care au atribute similare** (atributele fiind elemente de date, variabile interne, proprietati care caracterizeaza obiectele).

De exemplu, la intrarea intr-un laborator noi clasificam obiectele individuale: banci, studenti, si interactionam cu ele pe baza categoriei lor fara a fi necesar sa le cunoastem toate detaliile (atributele).

Clasa defineste elementele comune, numite in Java **CAMPURI** (**atribute** in teoria orientarii spre obiecte) si **METODE** (**operatii** in teoria orientarii spre obiecte), ale unei categorii de obiecte. Clasa reprezinta astfel **tipul de date al obiectelor**.

De exemplu, 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 un exemplu specific al unei clase, numit **instanta a clasei**, in care fiecare camp are o anumita valoare, iar **CLASA** este **tiparul dupa care sunt construite obiectele**.

De exemplu, o sala poate avea 30 de obiecte clasificate ca banci, fiecare banca avand propriile valori ale atributelor latime, inaltime, etc. Doua obiecte banca, *"banca1"* si *"banca2"*, sunt instante (exemple) diferite ale aceleiasi clase *"Banca"*, au in comun atributele, dar pot avea diferite valori ale lor:

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 apoi **numele 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

Atunci cand **modelam** bancile intr-un program de calcul putem sa cream spre exemplu doua banci (in limbajul Java) folosind urmatoarele portiuni de cod:

```
new Banca()
```

```
new Banca()
```

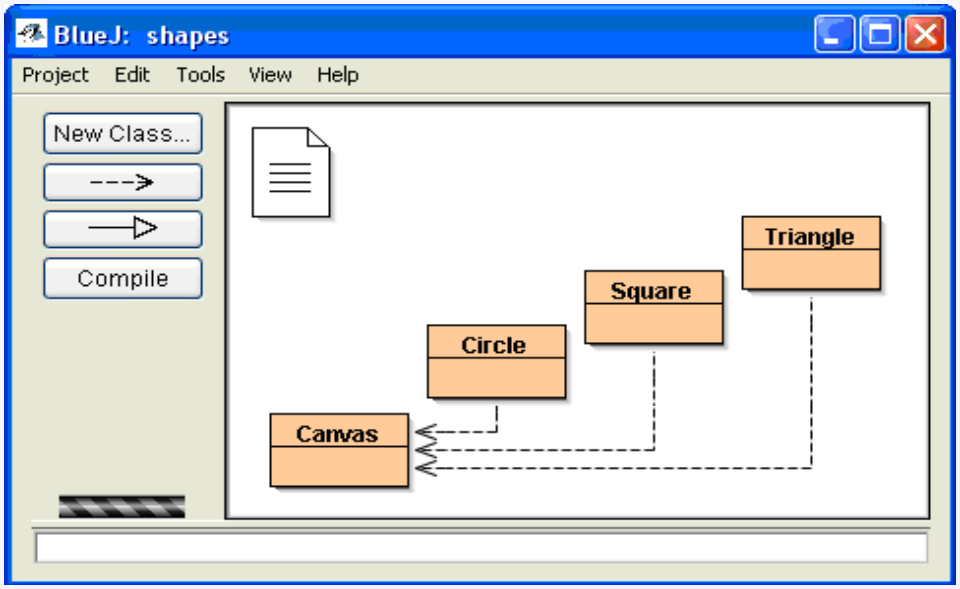
Pentru a putea trata (accesa) distinct cele doua obiecte, este necesara utilizarea a doua **nume diferite** pentru cele doua obiecte, ceea ce ar corespunde codului Java:

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

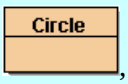
```
Banca banca2 = new Banca()
```

In laborator:
1. Dati nume cate unui obiect Java din fiecare dintre cele doua clase anterior numite.
2. Scrieti codul Java pentru crearea celor doua obiecte.

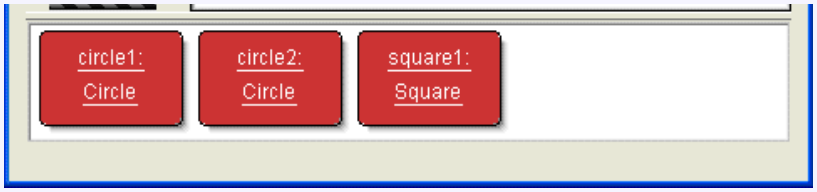
In laborator:
1. Lansati in executie mediul de dezvoltare **BlueJ**.
2. Deschideti proiectul numit ***shapes***.
   I. Click pe meniul **Project**, apoi selectati **Open Project …** (sau direct **Ctrl+O**)
   II. Selectati succesiv **C:**, **BlueJ**, **examples**, **shapes**, (sau scrieti **C:\BlueJ\examples\shapes**)



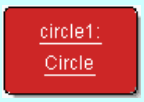
In laborator:

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

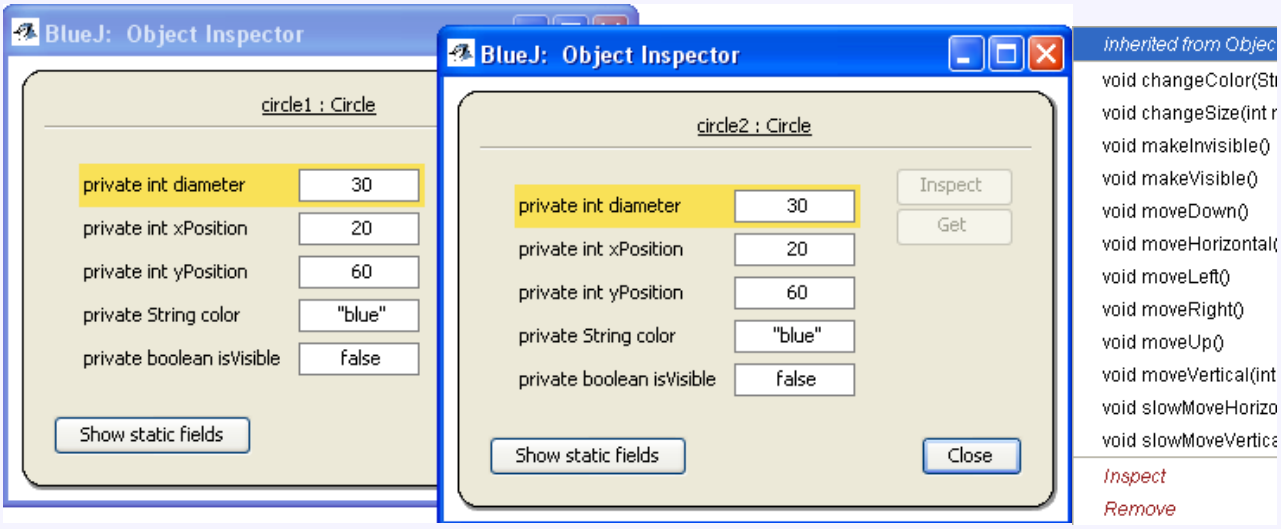
, selectati **new Circle()**, acceptati valoarea implicita.
2. Creati un alt cerc, acceptand din nou valoarea implicita oferita de BlueJ.
3. Creati un patrat (Square) in aceleasi conditii.



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*).



2.2.3. Apelul (invocarea) metodelor

Metoda Java (operatia in teoria OO), atunci cand este executata, realizeaza **o secventa de actiuni** (reprezentate in programe prin instructiuni) **asupra obiectului** caruia ii apartine. Actiunile realizate de executia metodelor au in general efect asupra valorilor campurilor (atributelor) obiectului.

Efectele acestor actiuni pot fi combinatii intre:

- **modificarea valorilor campurilor** obiectului, ca in cazul metodelor de tip **`setCamp()`**,
- **obtinerea valorilor campurilor**, ca in cazul metodelor de tip **`getCamp()`**,
- **realizarea altor sarcini utilizand aceste valori**.

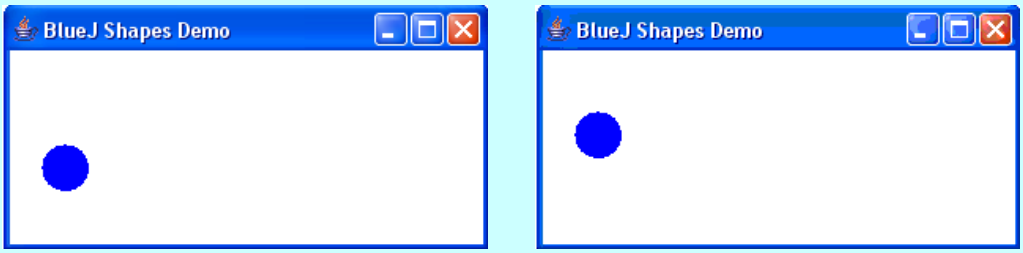
Regruparea mai multor elemente de date (campuri/atribute) si/sau de comportament (metode/operatii) asociate se numeste **incapsulare**. **Incapsularea OO** (orientata spre obiecte) inseamna in plus **ascunderea detaliilor interne** de tip:

- **informatii** (setul de **campuri/atribute**),
- si **implementare** (setul de **coduri interne ale metodelor/operatiilor**),

in spatele unei **interfete publice** (setul de **declaratii/semnaturi ale metodelor/operatiilor**).

In laborator:

1. Click-dreapta pe obiectul "`circle1`" si selectati **void makeVisible()**.
2. Click pe BlueJ Shapes Demo pentru a urmari efectul grafic al apelului metodei **makeVisible()**.
3. Click-dreapta pe obiectul "`circle1`" si selectati **moveUp()**. Urmariti efectul grafic.
4. Repetati apelul **moveUp()**, urmarind efectul grafic.



2.2.4. Parametrii metodelor

Parametrii specifica valorile de intrare necesare metodelor pentru a fi executate.

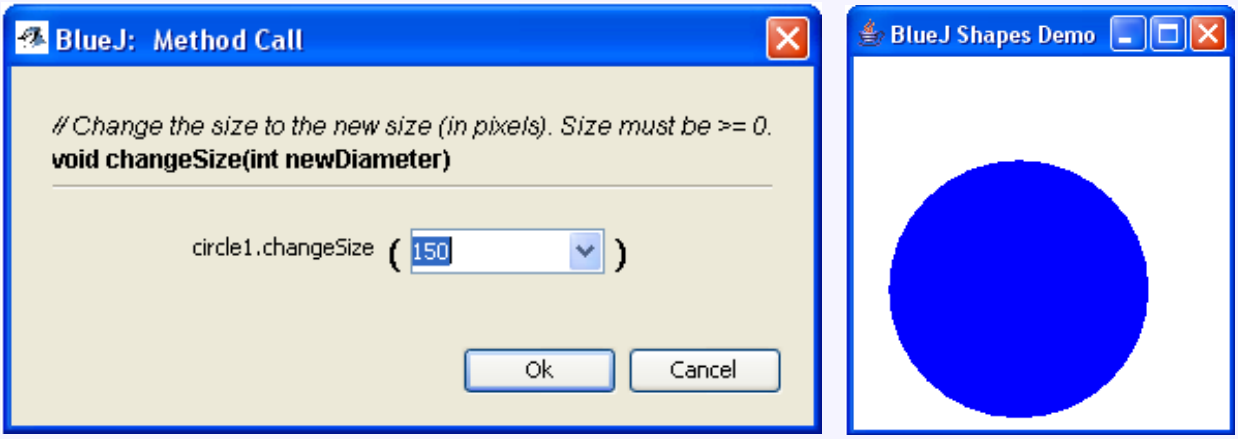
Declaratiile metodelor (semnaturile) pot include liste de declaratii de parametri. Acesti parametri sunt **variabile** care au ca scop intregul corp al metodei si se numesc **parametri formali** sau simplu **PARAMETRI**. Parametrii formali sunt declarati ca orice variabila, folosind formatul **`tipVariabila numeVariabila`**.

Apelurile metodelor pot include liste de valori date parametrilor, valori care trebuie sa corespunda ca tip celor declarate. Valorile pasatele metodelor in momentul apelurilor se numesc **parametri actuali** sau simplu **ARGUMENTE**.

De exemplu, apelul **`circle1.changeSize(50)`** specifica valoarea "`50`" ca argument, 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.



2.2.5. Tipuri de date

Descrierea problemelor reale sub forma de modele reprezentate ca programe 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
```

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

TIPUL DE DATE este o descriere abstracta a unui grup de entitati asemanatoare. Tipul de date defineste **structura variabilelor si domeniul de definitie al valorilor**.

Mai exact, tipul 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/multimea valorilor** posibile (-2^{31}…$2^{31}-1$ 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.) ;

- **regulile 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*, etc.) ;

- **operatorii asociati** (**permisi**) – care tin de partea de prelucrare asupra datelor.

Tipurile de date primitive Java:

Categorie	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	La char	La short, int, long, float, double
	short	0	16 biti (2B)	-32768 … 32767	La byte, char	La int, long, float, double
	int	0	32 biti (4B)	-2147483648 … 2147483647	La byte, short, char	La long, float, double
	long	0l	64 biti (8B)	-9223372036854775808 …9223372036854775807	La byte, short, int, char	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, char	La double
	double	0.0	64 biti (8B)	+/-4.9E-324 … +/-1.8+308, +/-infinity, +/-0, NaN	La byte, short, int, long, float, char	Nu exista (nu sunt necesare)
Caractere codificate UNICODE	char	\u0000 (null)	16 biti (2B)	\u0000 … \uFFFF	La byte, short	La int, long, float, double
Valori logice	boolean	false	1 bit folosit din 32 biti	true, false	Nu exista (nu sunt posibile)	Nu exista (nu sunt posibile)

In Java, pe langa tipurile de date primitive, exista si tipuri de date complexe numite **tipuri referinta**: **tablourile** si **clasele**.

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. 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?

2.2.6. Instante (obiecte) multiple

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

circle1		**circle2**	
int diameter	**30**	int diameter	**30**
int xPosition	**20**	int xPosition	**20**
int yPosition	**60**	int yPosition	**60**
String color	**"blue"**	String color	**"blue"**
boolean isVisible	**false**	boolean isVisible	**false**

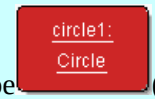
In laborator:

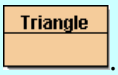
1. Creati trei obiecte "Circle".
2. Faceti fiecare obiect vizibil. Deplasati obiectele. Schimbati culorile obiectelor.

2.2.7. Starea unui obiect

Ansamblul valorilor campurilor (atributelor) unui obiect la un moment dat reprezinta **STAREA** obiectului. Starea unui obiect poate diferi in timp, ca urmare a **comportamentului**. Starea a doua obiecte de acelasi tip poate fi diferita la un moment dat.

In laborator:

1. Inspectati starea obiectului "`circle1`" cu double-click pe 

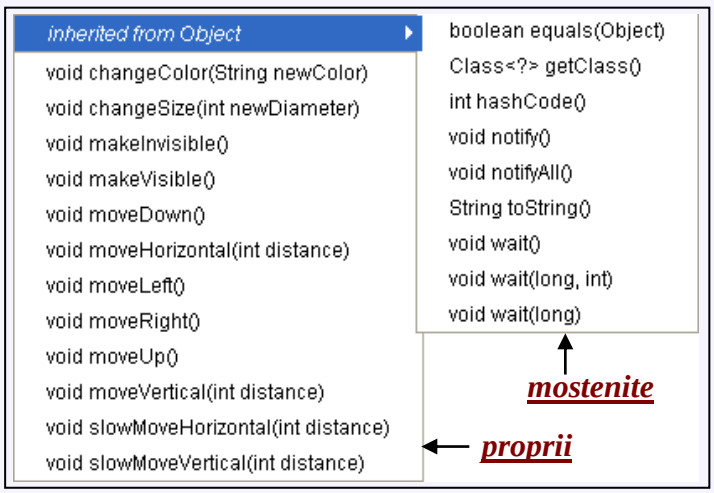
(sau click-dreapta si **Inspect**).
2. Schimbati culoarea obiectului "`circle1`" si inspectati-i din nou starea (valorile campurilor).
3. Creati doua obiecte 

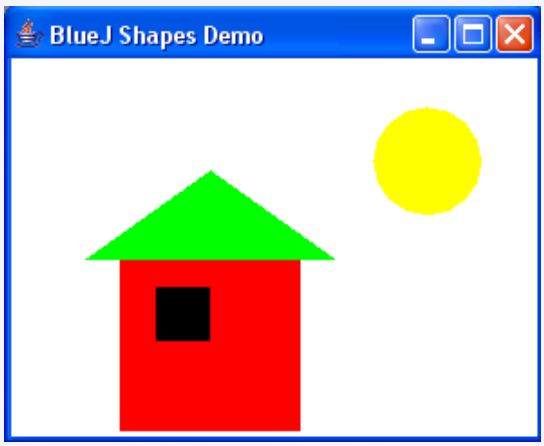
. Inspectati-le starea.
4. Au toate campurile aceleasi nume? Sunt toate valorile aceleasi?
5. Apelati metode care schimba pozitia celor doua obiecte. Inspectati-le starea. Ce s-a schimbat?
6. Creati doua obiecte din clase diferite. Inspectati-le starea. Ce campuri au aceleasi nume?

2.2.8. Comportamentul unui obiect

O **metoda** realizeaza o actiune asupra valorilor campurilor obiectului caruia ii apartine, putand folosi valorile acelor campuri, si astfel efectueaza o sarcina pentru codul (context) care a apelat-o. Metoda este un atom de **COMPORTAMENT** al obiectului. Comportamentul global al obiectului este obtinut prin inlantuirea apelurilor de metode. Toate obiectele din aceeasi clasa au aceleasi metode disponibile.

Clasa "`Circle`" are metodele:





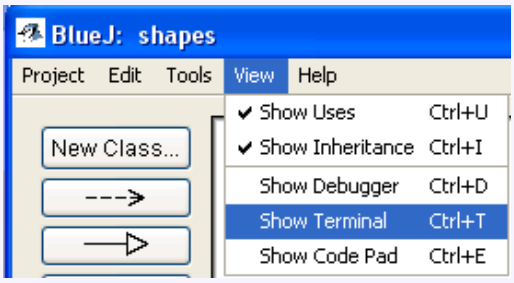
In laborator:

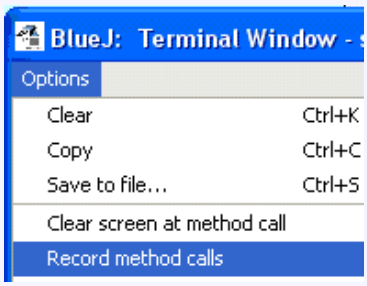
1. Creati o imagine care sa schiteze o casa si un soare similare celor din imaginea de mai sus.
2. Notati-va sarcinile pe care le-ati indeplinit pentru a obtine acest efect De exemplu:

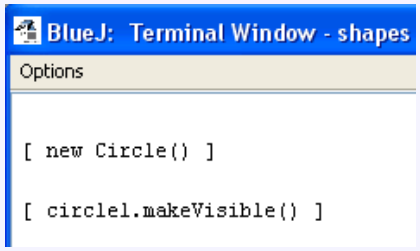
```
I.   Circle circle1 = new Circle()    // altfel spus, e creat un cerc
II.  circle1 makeVisible()            // apoi e facut vizibil cercul
III. circle1 moveHorizontal(200)      // e deplasat orizontal 200 pixeli
IV.  circle1 changeSize(50)           // e redimensionat la 50 pixeli
V.   circle1 changeColor("yellow")    // si e colorat in galben
VI.  ...
```

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

Observatie: Pentru a obtine automat pasii in forma electronica se deschide **Terminal Window** (cu **View-> Show Terminal** sau cu **Ctrl+T**) si se seteaza in acea fereastra **Options -> Record method calls**. In acest fel in **Terminal Window** vor fi scrisi automat pasii parcursi, ca in exemplul care urmeaza.







2.2.9. Interactiunea (colaborarea) 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** care creaza obiectele necesare si le apeleaza metodele, astfel incat ele sa fie pozitionate, dimensionate si colorate ca in desenul anterior. Obiectul de tip **Picture** interactioneaza (colaboreaza, comunica prin mesaje = apeluri metode) cu obiectele de tip Circle, Square si Triangle pentru a realiza sarcina globala.

In laborator:

1. Deschideti proiectul numit ***picture*** (**Ctrl-O**, apoi pe **C:\BlueJ\examples** selectati **picture**)
2. Creati un obiect de tip **Picture**.
3. Apelati metoda **void draw()**.
4. Click pe 

pentru urmari efectul grafic al apelului metodei.

2.2.10. Codul sursa Java. Editarea si compilarea cu BlueJ

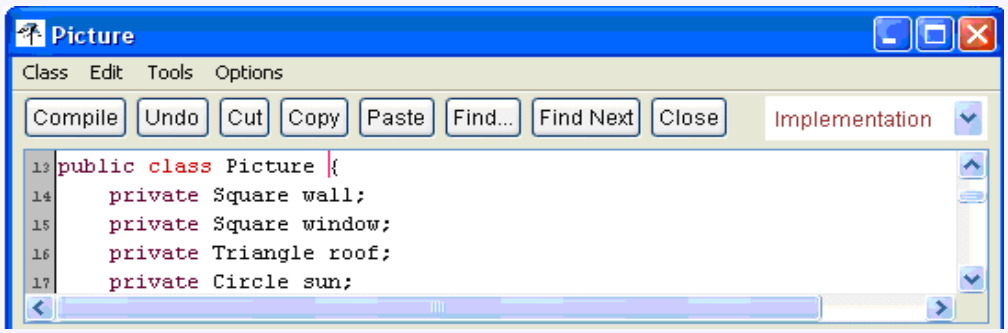
Sarcinile pentru crearea obiectelor si apelul metodelor pot fi scrise sub forma de instructiuni Java, salvate intr-un fisier, utilizate si reutilizate (executate) cand este nevoie de ele.

Listele instructiunilor Java (grupate in **metode**, iar acestea impreuna cu **campurile**) definesc o **clasa Java**. Textul scris al instructiunilor formeaza **codul sursa** al clasei. Pentru a fi executate, instructiunile trebuie mai intai **compilate** (translatate) cu compilatorul `javac` la **cod de octeti Java**. Apoi codul de octeti este **executat** de **interpretorul** `java`.

In laborator:

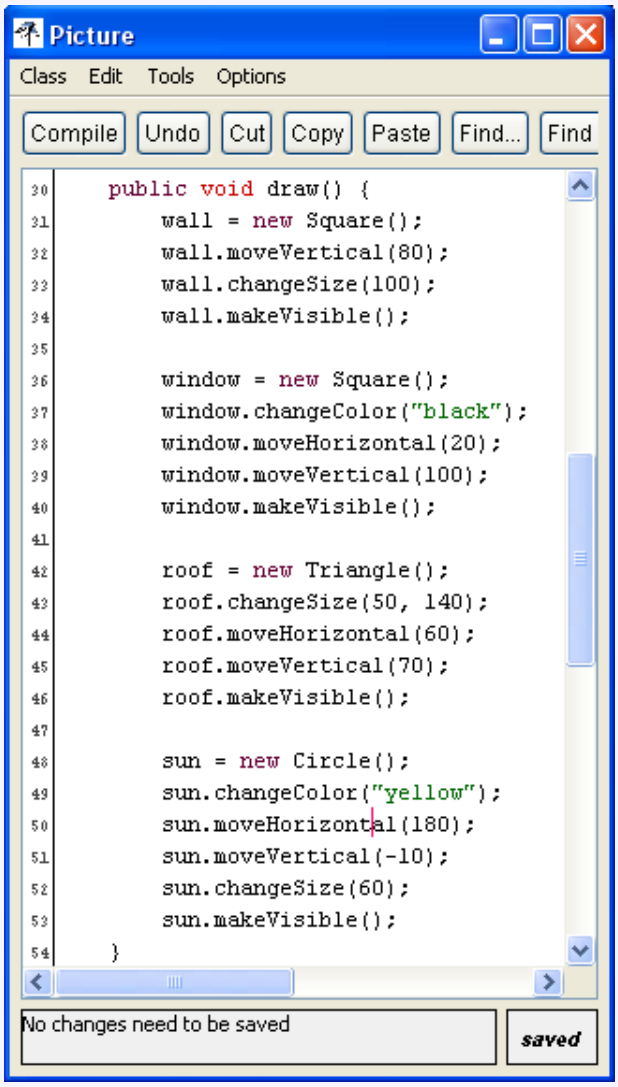
1. Deschideti proiectul numit ***picture***.
2. Vizualizati codul sursa al clasei **Picture**, fie prin double-click pe 

, 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.



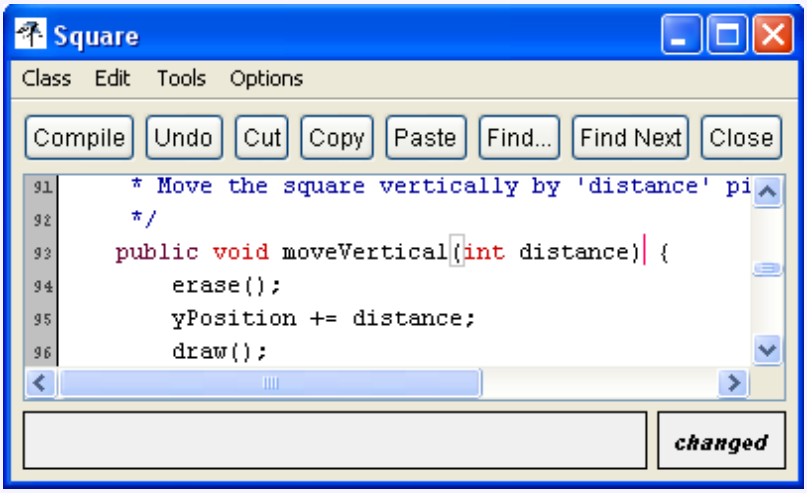
In laborator:

1. Gasiti codul metodei a carei declaratie (semnatura) este **public void draw()**.
2. Care sunt sarcinile elementare (instructiunile de tip apel) indeplinite pentru a crea zidul?
3. Conteaza ordinea in care sunt efectuate aceste sarcini?



In laborator:

1. Vizualizati codul sursa al clasei **Square**. Gasiti semnaturile metodelor invocate in metoda **draw()** a clasei **Picture**. Care sunt sarcinile indeplinite de codurile acestor metode?



2.3. Studiu de caz: Clasa care incapsuleaza informatii despre un student

2.3.1. Constructia structurii statice a clasei (doar campurile/atributele)

Sa presupunem ca dorim sa scriem codul unei clase Java numita **`Student`** care **sa abstractizeze un student real** (incapsuland informatii despre el) in cadrul unui program care gestioneaza informatii intr-o universitate, facultate, etc.

Pentru inceput ne vom concentra pe proprietatile (**atributele**, campurile Java) care constituie structura statica (datele, informatiile reprezentate sub forma de variabile) a clasei.

Pentru a selecta cateva informatii esentiale pentru modelul software al unui student putem mai intai sa ne imaginam care vor fi cazurile de utilizare (termen utilizat in limbajul UML pentru modelarea sistemelor software OO) ale clasei.

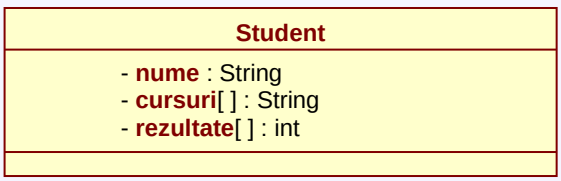
Acestea ar putea fi: *Inmatriculare, Repartizare in serie/grupa, Parcurgere semestru, Promovare semestru, Proiect de diploma, Cazare, Absolvire*.

Putem considera ca **esentiale** acele informatii care apar in mai multe astfel de cazuri de utilizare, cum ar fi ***numele*** studentului (informatie ce tine de persoana sa), ***cursurile / disciplinele*** pe care le are de parcurs (care tin de programa de studiu si de alegerile facute de el) si ***rezultatele / notele*** obtinute la aceste cursuri.

Reprezentand aceste informatii in forma cea mai simpla in limbajul Java rezulta codul:

```
1   /**
2    * Incapsuleaza informatiile despre un Student.
3    * @version 1.0
4    */
5   public class Student {
6     // Campuri (atribute) private (inaccesibile codurilor exterioare)
7     private String nume;       // nume + prenume intr-un singur sir de caractere
8     private String[] cursuri;  // numele cursurilor intr-un tablou de siruri
9     private int[] rezultate;   // notele asociate cursurilor intr-un tablou de int
10  }
```

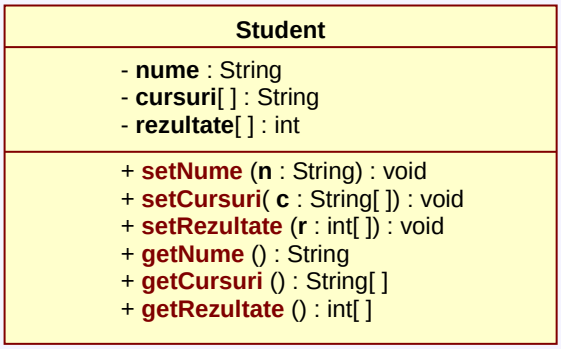
Reprezentarea UML (*Unified Modeling Language*) a **clasei** si **atributelor** (nivel de acces private, notat cu "-") este:



2.3.2. Adaugarea elementelor de comportament ale clasei (metodele/operatiile)

Pentru inceput ne vom concentra pe adaugarea **metodelor** necesare pentru accesul la proprietatile (atributele/campurile) care constituie structura clasei. Pentru fiecare camp Java vom adauga o metoda de modificare a valorii lui, denumita **`setCamp()`**, si o metoda de obtinere a valorii lui, **`getCamp()`**.

Reprezentarea UML a clasei, atributelor si metodelor (nivel de acces **public**, notat cu "+"):



Rezulta **codul Java**:

```
1   /**
2    * Incapsuleaza informatiile despre un Student.
3    * @version 1.1
4    */
5   public class Student {
6     // Campuri (atribute) private (inaccesibile codurilor exterioare)
7     private String nume;
8     private String[] cursuri;
9     private int[] rezultate;
10
11    // Metode (operatii) publice (accesibile tuturor codurilor exterioare)
12    // Metoda stabilire nume
13    public void setNume(String n)          {  nume = n;  }
14
15    // Metoda stabilire cursuri
16    public void setCursuri(String[] c)     {  cursuri = c;  }
17
18    // Metoda stabilire rezultate
19    public void setRezultate(int[] r)      {  rezultate = r;  }
20
21    // Metoda obtinere nume
22    public String getNume()                {  return (nume);  }
23
24    // Metoda obtinere cursuri
25    public String[] getCursuri()           {  return (cursuri);  }
26
27    // Metoda obtinere rezultate
28    public int[] getRezultate()            {  return (rezultate);  }
29  }
```

Informatii ascunse (campuri private)

Interfata publica (semnaturi ale metodelor)

Implementare ascunsa (coduri interne ale metodelor)

2.3.3. Adaugarea unei metode de test (metoda principala)

Pentru a putea testa codul clasei "`Student`" si lucrul cu obiectele este necesar sa adaugam o metoda principala (de test) in clasa "`Student`". Scenariul de test va contine:

- **crearea unui nou obiect** de tip `Student`,

- **initializarea campurilor** noului obiect, prin intermediul metodelor de tip **`getCamp()`**,

- **afisarea** numelui, disciplinei de **index 1** si a rezultatului asociat, folosind metodele **`setCamp()`**.

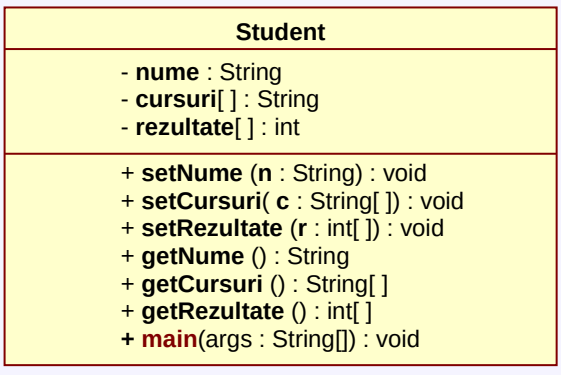
Rezulta codul Java:

```
1  /**
2     * Incapsuleaza informatiile despre un Student. Permite testarea locala.
3     * @version 1.2
4     */
5    public class Student {
6      // Campuri (atribute) private (inaccesibile codurilor exterioare)
7      private String nume;
8      private String[] cursuri;
9      private int[] rezultate;
10     // Metode (operatii) publice (accesibile tuturor codurilor exterioare)
11     public void setNume(String n) {       nume = n;  }
12     public void setCursuri(String[] c) {  cursuri = c;  }
13     public void setRezultate(int[] r) {   rezultate = r;  }
14     public String getNume() {             return (nume);  }
15     public String[] getCursuri() {        return (cursuri);  }
16     public int[] getRezultate() {         return (rezultate);   }
17     // Metoda de test. Punct de intrare in program.
18     public static void main(String[] args) {
19        // Crearea unui nou Student, fara informatii
20        Student st1 = new Student();
21        // Initializarea campurilor noului obiect
22        st1.setNume("Xulescu Ygrec");
23        String[] crs = {"CID", "AMP", "MN"};
24        st1.setCursuri(crs);
25        int[] rez = {8, 9, 10};
26        st1.setRezultate(rez);
27        // Utilizarea informatiilor privind Studentul
28        System.out.println("Studentul " + st1.getNume() + " are nota "
29          + st1.getRezultate()[1] + " la disciplina " + st1.getCursuri()[1]);
30     }
31   }    // Rezultatul:  Studentul Xulescu Ygrec are nota 9 la disciplina AMP
```

In laborator:

1. Inchideti proiectele anterioare (cu **Project** si **Close** sau **Ctrl+W**).
2. **Creati un nou proiect** numit ***StudProject*** (cu **Project**, apoi **New Project…**, selectati calea de tip `D:\LPAI\Grupa_44#\SubgrupaX`, si scrieti ***StudProject***).
3. Creati o noua clasa, numita `Student`, apasand **New Class…**
4. Double-click pe noua clasa (ii deschideti codul in editor), si scrieti codul de mai sus.
5. Compilati codul sursa prin click pe butonul **Compile** si executati metoda main() a noii clase (right-click pe clasa si selectare **main()**).

Reprezentarea UML actualizata a clasei, atributelor si metodelor:



2.3.4. O alternativa: metoda de test intr-o clasa separata (clasa de test)

O alternativa la versiunea anterioara este utilizarea unei **clase distincte** pentru testarea clasei "Student", numita "TestStudent", care sa contina doar o **metoda principala** pentru ilustrarea lucrului cu obiectele clasei "Student" (scenariul de test afisand disciplina de **index 0** si rezultatul asociat).

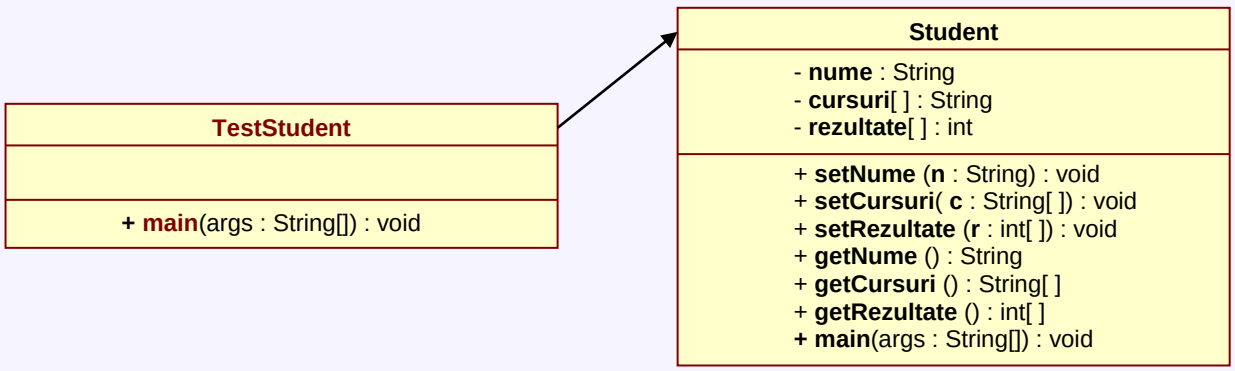
```
1   /**
2    * Testeaza clasa Student version 1.1 and 1.2.
3    */
4   public class TestStudent {
5     // Metoda de test. Punct de intrare in program.
6     public static void main(String[] args) {
7        // Crearea unui nou Student, fara informatii
8        Student st1 = new Student();
9        // Initializarea campurilor noului obiect
10       st1.setNume("Xulescu Ygrec");
11       String[] crs = {"CID", "AMP", "MN"};
12       st1.setCursuri(crs);
13      int[] rez = {8, 9, 10};
14       st1.setRezultate(rez);
15       // Utilizarea informatiilor privind Studentul
16       System.out.println("Studentul " + st1.getNume() + " are nota "
17         + st1.getRezultate()[0] + " la disciplina " + st1.getCursuri()[0]);
18    }
19  }    // Rezultatul:  Studentul Xulescu Ygrec are nota 8 la disciplina CID
```

In laborator:

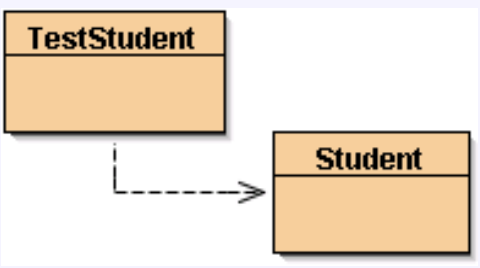
1. Tot in proiectul *StudProject*, creati o noua clasa numita **`TestStudent`**
2. Double-click pe noua clasa (deschideti editorul) si scrieti codul cu cel de sus.
3. **Compilati** codul si **executati** metoda **main()** a noii clase.

Desi clasa "Student" in versiunea 1.2 contine metoda **`main()`**, ea poate fi testata si folosind clasa "TestStudent". Astfel, noua clasa poate testa atat versiunea 1.1 cat si versiunea 1.2 a clasei "Student".

Reprezentarea UML a claselor (simplificata) si a asocierii intre clase (notata cu o sageata) este:



In BlueJ:



2.4. Teme pentru acasa

Tema de casa pentru data viitoare (temele vor fi predate la lucrarea urmatoare**,** pe hartie**,** scrise de mana): **Codurile sursa** ale unei clase create dupa modelul clasei "`Student`", sectiunea 2.3, cu urmatoarea specificatie generala:

- **clasa** cu numele alocat din tabelul care urmeaza (numita generic **`x`**),

- va avea **4 campuri (atribute)** cu acces **`private`**, considerate esentiale (de catre autor) pentru clasa respectiva,

- **fiecare camp va avea cate doua metode cu acces `public`**, una de tip **`get...()`** prin care va fi obtinuta valoarea campului, si una de tip **`set...()`**, prin care va fi stabilita valoarea campului,

- va exista o **metoda principala**, **scenariul de test** din metoda principala:
  - va crea un nou obiect din clasa **`x`**,
  - va initializa campurile noului obiect folosind metodele de tip **`set...()`**,
  - va afisa valorile campurilor obiectului obtinute cu metodele de tip **`get...()`**.

Fiecare grup de cate doi studenti (grupuri stabilite in timpul desfasurarii acestui laborator) va avea alocate doua nume de clasa asemanatoare, conform tabelului:

Nr. ordine grup	Numele claselor (pentru tema de casa)	Nr. ordine grup	Numele claselor (pentru tema de casa)
1	`Monitor + Televizor`	**5**	`Produs+Serviciu`
2	`Masina + Camion`	**6**	`PC + Laptop`
3	`Proiector+ Smart whiteboard`	**7**	`Veioza+Lanterna`
4	`Aparat Foto+Camera Video`	**8**	`Stick USB+CD-ROM`

Membrii fiecarui grup vor stabili intre ei ce clasa alege fiecare pentru a realiza tema (un student va alege o clasa iar al doilea student cealalta clasa). Deoarece clasele alocate unui grup sunt asemanatoare, membrii grupului vor alege 2 atribute comune celor doua clase si 2 atribute distincte pentru fiecare dintre clase.

Metoda principala va fi parte a clasei alese pentru un membru al grupului si parte a unei clase de test pentru celalalt membru.

Anexa. Instalarea kitului BlueJ

1. Se lanseaza
   bluejsetup-303.exe (http://www.bluej.org/download/files/bluejsetup-303.exe)
2. Se confirma TOATE optiunile implicite!!
3. In final, se deselecteaza **"View the README file"** si se selecteaza **"Launch BlueJ"**
4. Se confirma versiunea Java **C:\Program Files\Java\jdk1.5.0_12**
   (NU ALTA, daca cumva exista pe calculator) si se apasa Launch BlueJ
5. Pentru testarea instalarii se selecteaza **Project -> Open Project** (sau direct **Ctrl-O**)
6. Se selecteaza pe rand C: apoi BlueJ apoi examples si in final shapes
   (calea va fi **C:\BlueJ\examples\shapes**)
7. Se selecteaza cu Ctrl-A toate clasele si apoi se apasa Compile
   (dreptunghiurile ar trebui sa devina clare din hasurate)
8. Cu buton dreapta pe **Circle** se selecteaza **new Circle()** si se urmareste efectul pe ecran.