POO – Laborator

Modificarea programelor Java. Studiu de caz: programul Salut.java

1. Varianta care utilizeaza un argument al programului si o variabila locala

O prima varianta de program, **`SalutPrimulArgument.java`**, **utilizeaza un argument dat programului de catre utilizator, la lansare** (pentru a pasa programului numele care trebuie salutat), **si o variabila locala**.

```
 1   /**
 2    *  Exemplifica constructii de baza ale limbajului Java. Clasa principala.
 3    */
 4   public class SalutPrimulArgument {
 5     /**
 6      *  Afiseaza un salut catre utilizator. Metoda principala.
 7      *  @param args Lista argumentelor programului.
 8      */
 9     public static void main (String[] args) {
10
11       // Variabila locala
12       String variabilaLocala = "Salut ";
13
14       // Utilizarea unui argument si al unei variabile locale
15       // Concatenare de siruri de caractere
16       System.out.println(variabilaLocala + args[0]);
17     }
18   }
```

Argumentele sunt furnizate de catre utilizator la lansarea programelor in executie **din linia de comanda**, sub forma:

directorcurent> **`java <NumeClasa> <argumente program java>`**

Argumentele sunt despartite prin spatii, si sunt accesate in interiorul programului sub forma de siruri de caractere (obiecte ale clasei **`String`**). Daca se doreste ca mai multe cuvinte sa formeze un singur argument, acestea sunt plasate intre ghilimele. In cazul clasei **`SalutPrimulArgument`**, primul argument (**`args[0]`**, elementul de index **`0`** al tabloului argumentelor) va fi numele utilizatorului, programul urmand sa il salute pe utilizator.

Daca utilizatorul foloseste comanda:

directorcurent> **`java SalutPrimulArgument Nick`**

efectul este:

`Salut Nick`

Acelasi efect se obtine daca utilizatorul foloseste comanda:

directorcurent> **`java SalutPrimulArgument Nick Name`**

Pentru a se obtine efectul:

`Salut Nick Name`

utilizatorul trebuie sa foloseasca comanda:

directorcurent> **`java SalutPrimulArgument "Nick Name"`**

Programul foloseste o variabila locala (numita **`variabilaLocala`**) pentru stocarea unei parti din mesajul catre utilizator (**`"Salut "`**).

Variabilele locale sunt acele variabile **declarate in interiorul blocurilor de cod** (cuprinse intre acolade), sunt **create in momentul declararii lor** si sunt **accesibile din locul declararii si pana la sfarsitul blocului de cod in care au fost declarate** (pentru variabilele referinta in momentul declararii este creata doar referinta, obiectul referit fiind creat dinamic cu **`new`**).

Concatenarea sirurilor de caractere `variabilaLocala` si **`args[0]`** este realizata cu ajutorul **operatorului "+".**

2. Varianta care citeste consola standard de intrare (tastatura) si utilizeaza un atribut static

O alta varianta de program, **`SalutUtilizator1.java`**, **obtine numele caruia i se adreseaza salutul prin intermediul consolei standard de intrare** (din fereastra de comanda) si **utilizeaza un atribut**.

```
 1   /**
 2    *  Importul pachetului de clase de intrare-iesire (IO).
 3    */
 4   import java.io.*;
 5   /**
 6    *  Exemplifica constructii de baza ale limbajului Java. Clasa principala.
 7    */
 8   public class SalutUtilizator1 {
 9     /**
10      *  Numele utilizatorului. Atribut global (static),
11      *  partajat de clasa si de toate obiectele clasei.
12      */
13     public static String atribut;
14     /**
15      *  Afiseaza un salut catre utilizator. Metoda principala.
16      */
17     public static void main(String[] args) throws IOException {
18       BufferedReader intrareConsola =
19             new BufferedReader(new InputStreamReader(System.in));
20       System.out.print("Introduceti-va numele: ");
21       atribut = intrareConsola.readLine();
22       System.out.println("Buna ziua " + atribut + "!");
23     }
24   }
```

Citirea consolei standard de intrare se face prin intermediul unui flux de intrare pentru caractere (in Java, caracterele sunt codate UNICODE pe 2 octeti), declarat in linia 20, **conectat la fluxul standard de intrare (care ofera datele de la tastatura sub forma de octeti simpli) prin intermediul unui flux de intrare care face conversia octet-caracter** (linia 21).

Dupa lansare, programul afiseaza mesajul **`"Introduceti-va numele: "`**, iar daca utilizatorul introduce numele **`Nick`** urmat de **`<Enter>`**, efectul este urmatorul:

```
directorcurent> java SalutUtilizator1
Introduceti-va numele: Nick
Buna ziua Nick !
```

Atributul (numit si variabila membru a clasei) este o variabila declarata la acelasi nivel cu metodele clasei (functiile ei membru), este creata in momentul crearii clasei (daca este declarata **`static`**) sau in momentul crearii obiectului (daca **nu este declarata `static`**), si este accesibila in

interiorul clasei, si inainte si dupa locul declararii ei (ba chiar si in exteriorul clasei, in functie de specificatorul de acces folosit).

Atributul folosit aici, numit **`atribut`**, este **declarat `static`** (este global, fiind o locatie unica la nivelul clasei) si **`public`** (este accesibil de catre orice alta clasa Java).

Daca ar fi fost **declarat fara `static`** (local, fiind creata o locatie separata pentru fiecare obiect) programul ar fi trebui modificat pentru a putea functiona (**se poate verifica efectul indepartarii cuvantului cheie `static`**). Pentru a functiona fara **`static`**, in locul liniilor 22 si 23 ar fi trebuit scris:

```
22        SalutUtilizator1 salut = new SalutUtilizator1();
23        salut.atribut = intrareConsola.readLine();
24        System.out.println("Buna ziua " + salut.atribut + "!");
```

3. Varianta care utilizeaza o fereastra de dialog, un atribut non-static, un constructor, o metoda si un obiect al clasei curente

O alta varianta de program, **`SalutUtilizator2.java`**, **obtine numele caruia i se adreseaza salutul prin intermediul unei** fereastre de dialog (Swing).

```
 1    /**
 2     *  Clasa importata din pachetul de clase grafice avansate (swing).
 3     */
 4    import javax.swing.JOptionPane;
 5    /**
 6     *  Exemplifica constructii de baza ale limbajului Java. Clasa principala.
 7     */
 8    public class SalutUtilizator2 {
 9      /**
10       *  Numele utilizatorului.
11       *  Atribut local (non-static), la nivel de obiect, cu acces public.
12       */
13      public String nume;
14      /**
15       *  Initializeaza atributul obiectului curent. Constructor.
16       */
17      public SalutUtilizator2() {
18        // Obtinerea numelui utilizatorului prin fereastra de dialog
19        nume = JOptionPane.showInputDialog("Introduceti-va numele:");
20      }
21      /**
22       *  Afiseaza un salut. Metoda utilitara.
23       */
24      public void salutaNume() {
25        // Concatenare de siruri de caractere
26        System.out.println("Hello " + nume + "!");
27      }
28      /**
29       *  Afiseaza un salut catre utilizator. Metoda principala.
30       */
31      public static void main(String[] args) {
32        // Variabila locala initializata cu un nou obiect afisor
33        SalutUtilizator2 afisor = new SalutUtilizator2();
34
35        // Apelul metodei de afisare a numelui
36        afisor.salutaNume();
37      }
38    }
```

Fereastra de dialog este cel mai simplu si elegant **mod de obtinere a unei intrari de la utilizator** atunci cand este nevoie. Pentru a fi utilizata este necesara declaratia de import din linia **4** si apelul din linia **19**.

Metodei **`showInputDialog()`** i se ofera ca parametru mesajul catre utilizator, iar metoda returneaza intrarea de la utilizator sub forma de sir de caractere (obiect ale clasei **`String`**).

Atributul non-static **`nume`** impune crearea unui obiect al clasei **`SalutUtilizator2`** (linia 33) pentru a putea fi utilizat.

Constructorii sunt metode care **au acelasi nume cu al clasei**, **nu returneaza valori**, sunt **apelate in momentul crearii obiectelor** (dupa operatorul **`new`**) si sunt **folosite in general la initializarea atributelor non-statice.**

Constructorul fara parametri **`SalutUtilizator2()`** initializeaza atributul **`nume`.**

Constructorul este un **caz particular de delegare a functionalitatii**, in acest caz fiind delegata initializarea atributelor si a altor resurse in momentul crearii obiectelor.

Metodele (functiile membru) sunt singurele forme in care pot fi declarate functiile in Java. Ele pot fi **cu sau fara parametri**, si **pot returna sau nu valori** (in acest ultim caz, tipul valorii returnate este `void`). Metoda **`salutaNume()`** nu are parametri si afiseaza un mesaj care contine atributul **`nume`.**

Metodele sunt **cazul cel mai simplu si mai des intalnit de delegare a functionalitatii**. Separarea codului in functii (metode) serveste gestiunii mai simple a codului, modularizarii codului la nivel de secventa de instructiuni, reutilizarii codului la nivel de functie.

In general **functia principala** **`main()`** **deleaga anumite sarcini** catre alte functii, **acestea la randul lor** catre altele, s.a.m.d.

O forma mai complexa si mai avansata de delegare a functionalitatii este **separarea codului in mai multe clase si obiecte** (fiecare la randul ei separata functional).

In acest caz **metoda principala din clasa principala deleaga anumite sarcini** catre alte metode ale clasei principale (mai rar) si catre metode ale altor clase. **Acestea la randul lor pot delega sarcini** catre alte metode ale acelorasi clase sau catre metode ale altor clase, s.a.m.d.

4. Varianta care utilizeaza delegarea la nivel de clase

O alta varianta de program, **`SalutUtilizator3.java`**, utilizeaza :

- delegarea functionalitatilor de obtinere a informatiilor catre o noua clasa, utilitara, numita **`Obtinere`**, si

- delegarea functionalitatilor de afisare a informatiilor catre o noua clasa, utilitara, numita **`Afisare`**.

Clasa **`SalutUtilizator3`** are rolul de **dispecer de sarcini**, metoda ei principala continand **scenariul principal** (citirea prenumelui de la consola, afisarea unui salut in consola, citirea numelui prin fereastra grafica, afisarea unui salut in fereastra grafica).

Ambele clase utilitare ofera atat varianta consolei standard cat si cea a ferestrelor de dialog. Metodele lor sunt de tip **`static`**.

```
1   import javax.swing.JOptionPane;
2
3   /**
4    *  Exemplifica constructii de baza ale limbajului Java. Clasa utilitara.
5    */
6   public class Afisare {
7     /**
8      *  Afiseaza un text in consola. Metoda utilitara.
9      *  @param text Textul de afisat.
10     */
11    public static void textInConsola(String text) {
12      System.out.println(text);
13    }
14
15    /**
16     *  Afiseaza un text in fereastra de dialog. Metoda utilitara.
17     *  @param text Textul de afisat.
18     */
19    public static void textInFereastra(String text) {
20      JOptionPane.showMessageDialog(null, text);
21    }
22  }
```

```
1   import javax.swing.JOptionPane;
2
3   /**
4    *  Importul pachetului de clase de intrare-iesire (IO).
5    */
6   import java.io.*;
7   /**
8    *  Exemplifica constructii de baza ale limbajului Java. Clasa utilitara.
9    */
10  public class Obtinere {
11    /**
12     *  Obtine un text prin consola. Metoda utilitara.
13     *  @param mesaj Mesajul catre utilizator.
14     *  @return Textul obtinut.
15     */
16    public static String textDinConsola(String mesaj) throws IOException {
17      BufferedReader inConsola
18            = new BufferedReader(new InputStreamReader(System.in));
19      System.out.print(mesaj);
20      String text = inConsola.readLine();
21      return (text);
22    }
23
24    /**
25     *  Obtine un text printr-o fereastra grafica. Metoda utilitara.
26     *  @param mesaj Mesajul catre utilizator.
27     *  @return Textul obtinut.
28     */
29    public static String textDinFereastra(String mesaj) {
30      String text = JOptionPane.showInputDialog(mesaj);
31      return (text);
32    }
33  }
```

```
 1   /**
 2    *  Clasa importata din pachetul de clase grafice avansate (swing).
 3    */
 4   import javax.swing.JOptionPane;
 5   /**
 6    *  Importul pachetului de clase de intrare-iesire (IO).
 7    */
 8   import java.io.*;
 9   /**
10    *  Exemplifica constructii de baza ale limbajului Java. Clasa principala.
11    */
12   public class SalutUtilizator3 {
13     /**
14      *  Afiseaza un salut catre utilizator. Metoda principala.
15      *  @param args Lista argumentelor programului.
16      */
17     public static void main(String[] args) throws IOException {
18       // Variabila locala
19       String prenume = Obtinere.textDinConsola("Introduceti-va prenumele:");
20
21       // Apelul metodei de afisare a numelui
22       Afisare.textInConsola("Buna ziua, " + prenume + "!");
23
24       // Variabila locala
25       String nume = Obtinere.textDinFereastra("Introduceti-va numele:");
26
27       // Apelul metodei de afisare a numelui intreg
28       Afisare.textInFereastra("Buna ziua, " + prenume + " " + nume + "!");
29     }
30   }
```

De reflectat:

1. Cum ar arata programul `SalutUtilizator3` daca metodele claselor `Obtinere` si `Afisare` **ar fi non-statice?**

2. Cum trebuie modificat programul `SalutUtilizator3` pentru a putea executa in mod repetat scenariul anterior (citire prenume, afisare salut in consola, citire nume, afisare salut in fereastra)?