ISw - Seminar 2.

Lucrul cu variabile de tip primitiv si referinta (II)

2.1. Descrierea seminarului

In acest seminar vor fi acoperite total sau partial urmatoarele probleme:

- implementarea constructorilor si metodelor,

- modificarea constructorilor si metodelor,

- supraincarcarea (*overloading*) numelor constructorilor si metodelor,

- extinderea claselor prin **mostenire**,

- rescrierea (*overriding*) metodelor,

- compararea a doua variabile de tip primitiv, a **doua referinte** (referinte catre obiecte diferite sau catre acelasi obiect), a **continuturilor a doua obiectelor** (obiecte diferite cu continut diferit sau identic).

2.2. Probleme rezolvate si probleme propuse spre rezolvare

Problema 2.1.

a) Sa se scrie un program Java, numit **`Punct.java`**, **care sa defineasca o clasa `Punct`**, avand:

- **atributele**: **`x`** de tip **`int`**, **`y`** de tip **`int`**, si **`numePunct`** de tip **`String`**,
- un **constructor** (metoda cu caracter special, care are acelasi nume cu al clasei, nu returneaza valori si e utilizata pentru initializarea atributelor obiectului in momentul instantierii lui) cu declaratia:

```
public Punct(int a, int o, String id)
```

care sa initializeze atributele **`x`**, **`y`**, si **`numePunct`**, cu valorile parametrilor **`a`**, **`o`** si respectiv **`id`**.

- o **metoda cu caracter de obiect** (fara cuvantul cheie `static`) numita **`afisarePunct()`** care nu primeste nici un parametru, nu returneaza nici o valoare, si afiseaza continutul atributelor in formatul:

```
Punctul P (2, 1)
```

presupunand ca **`x`**, **`y`**, si **`numePunct`** au valorile **`2`**, **`1`** si respectiv **`"P"`**.

- **metoda principala** care foloseste constructorul pentru a initializa atributele **`x`**, **`y`**, si **`numePunct`** cu valorile **`2`**, **`1`** si respectiv **`"P"`** si metoda **`afisarePunct()`**pentru a afisa continutul atributelor clasei.

Rezolvare:

O posibila rezolvare este urmatoarea:

```
1    public class Punct {
2      protected int x;
3      protected int y;
4      protected String numePunct;
5
6      public Punct(int a, int o, String id) {
7        x = a;
8        y = o;
9        numePunct = id;
10     }
11     public void afisarePunct() {
12       System.out.println("Punctul " + numePunct +
13                  " (" + this.x + ", " + this.y + ")");
14     }
15     public static void main(String[] args) {
16       Punct p = new Punct(2, 1, "P");
17       p.afisarePunct();
18     }
19   }
```

b) Sa se modifice programul `Punct.java`:

- metoda **`afisarePunct()`** urmand sa afiseze continutul returnat de metoda **`toString()`** (mostenita de la clasa **`Object`**) in formatul:

```
Punctul <valoarea returnata de metoda toString()>
```

Rezolvare:

Orice clasa Java care nu extinde (prin mostenire) in mod explicit o alta clasa Java, **extinde** (prin mostenire) in mod **implicit clasa `Object`** (radacina ierarhiei de clase Java), clasa care contine metodele necesare tuturor obiectelor Java.

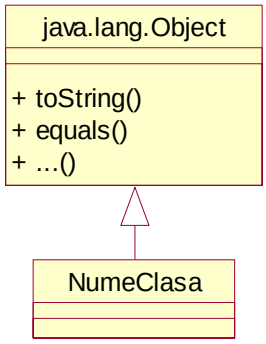
Urmatoarea **declaratie de clasa**:

```
class NumeClasa { // urmeaza corpul clasei ...
```

este **echivalenta cu**:

```
class NumeClasa extends Object { // urmeaza corpul clasei ...
```

Diagrama UML corespunzatoare **codului Java** anterior:



Printre metodele declarate in clasa **`Object`** este si **`toString()`**, metoda care are ca **scop returnarea sub forma de `String` a informatiilor pe care le incapsuleaza obiectul caruia i se aplica aceasta metoda**.

In **cazul claselor scrise de programator, in mod implicit** metoda **`toString()`** returneaza numele clasei careia ii apartine obiectul urmat de un cod alocat acelui obiect (*hashcode*). Implementarea implicita a metodei **`toString()`** este urmatoarea:

```
1    // Implementarea implicita a metodei toString(),
2    // mostenita de la clasa Object
3
4    public String toString() {
5       // (nu returneaza continutul ci numele clasei si codul obiectului!)
6       return getClass().getName() + "@" + Integer.toHexString(hashCode());
7    }
```

In aceste conditii, **o posibila rezolvare** este urmatoarea:

```
 1    public class Punct {
 2      protected int x;                              // atribute
 3      protected int y;
 4      protected String numePunct;
 5
 6      public Punct(int a, int o, String id) {       // constructor
 7        x = a;
 8        y = o;
 9        numePunct = id;
10      }
11      public void afisarePunct() {                  // metoda non-statica
12        System.out.println("Punctul " + this.toString()); // cod mostenit
13      }
14      public static void main(String[] args) {      // metoda principala
15        Punct p = new Punct(2, 1, "P");
16        p.afisarePunct();
17      }
18    }
```

Apelul **`this.toString()`** va returna un sir de caractere de forma "**`Punct@1add2dd`**", format prin concatenarea numelui clasei, "**`Punct`**", cu caracterul "**`@`**", si cu codul "**`1add2dd`**" (*hash code*).

c) Sa se modifice programul `Punct.java`, adaugand o noua implementare metodei `toString()`, astfel incat metoda **`afisarePunct()`** sa afiseze informatiile incapsulate in obiectul de tip **`Punct`** in formatul:

```
Punctul P (2, 1)
```

presupunand ca **`x`**, **`y`**, si **`numePunct`** au valorile **`2`**, **`1`** si respectiv "**`P`**".

Rezolvare:

In **cazul claselor de biblioteca Java**, metoda **`toString()`** returneaza ansamblul valorilor curente ale atributelor obiectului.

In **cazul in care programatorul doreste returnarea informatiilor incapsulate in obiect, trebuie specificat in mod explicit un nou cod (o noua implementare)** pentru metoda **`toString()`**. Acest lucru se obtine adaugand clasei din care face parte acel obiect o metoda cu declaratia:

```
public String toString() { // urmeaza corpul metodei ...
```

metoda care se spune ca **rescrie (*overrides*)** codul metodei cu acelasi nume din clasa extinsa (in acest caz clasa **`Object`**).

Dupa adaugarea acestei metode, apelul **`toString()`** (sau **`this.toString()`**) va conduce la executia noului cod, pe cand apelul **`super.toString()`** va conduce la executia codului din clasa extinsa (in acest caz codul implicit din clasa **`Object`**).

O posibila rezolvare este urmatoarea:

```
1    public class Punct {
2      protected int x;
3      protected int y;
4      protected String numePunct;
5
6      public Punct(int a, int o, String id) {
7        x = a;
8        y = o;
9        numePunct = id;
10     }
11     public void afisarePunct() {
12       System.out.println("Punctul " + this.toString());       // cod nou
13     }
14     public String toString() {
15       return numePunct + " (" + this.x + ", " + this.y + ")"; // cod nou
16     }
17     public static void main(String[] args) {
18       Punct p = new Punct(2, 1, "P");
19       p.afisarePunct();
20     }
21   }
```

Apelul **`this.toString()`** va returna acum un sir de caractere de forma "**`P (2, 1)`**", format prin concatenarea atributului nume, "**`P`**", cu caracterele " (", cu valoarea abscisei, "**`2`**", cu caracterele ", ", cu valoarea ordonatei, "**`1`**", si cu caracterul ")".

Problema 2.2.

Sa se creeze o noua clasa `PunctExtins`, care **sa extinda (prin mostenire) clasa `Punct`**, prin declararea in mod explicit:

- a unui **constructor** cu declaratia:

```
public PunctExtins(int a, int o, String id) {
```

care sa initializeze atributele **`x`**, **`y`**, si **`numePunct`**, cu valorile parametrilor **`a`**, **`o`** si respectiv **`id`**, prin intermediul constructorului clasei pe care o extinde (**`Punct`**).

- a unei **metode cu caracter de obiect** (fara cuvantul cheie `static`) numita **`equals()`** care primeste ca parametru un obiect al clasei **`Object`**, compara obiectul primit cu obiectul curent si returneaza valoare de tip **`boolean`** corespunzatoare rezultatului comparatiei.

```
public boolean equals(Object obj) {
```

Rezolvare:

Printre metodele declarate in clasa **`Object`** este si **`equals()`**, metoda care are ca scop **compararea obiectului caruia i se aplica aceasta metoda cu un obiect pasat ca parametru**, returnand valoarea booleana **`true`** in cazul egalitatii si valoarea booleana **`false`** in cazul inegalitatii celor doua obiecte.

In **cazul claselor de biblioteca Java**, metoda **`equals()`** compara ansamblul valorilor curente ale atributelor obiectului (continutul sau starea obiectului).

In **cazul claselor scrise de programator, in mod implicit** metoda **`equals()`** compara referinta obiectului caruia i se aplica aceasta metoda cu referinta obiectului pasat ca parametru. Implementarea implicita a metodei **`equals()`** este urmatoarea:

```
1    // Implementarea implicita a metodei equals(),
2    // mostenita de la clasa Object
3
4    public boolean equals(Object obj) {
5      return (this == obj);     // (nu compara continutul ci referintele!!!)
6    }
```

In **cazul in care programatorul doreste compararea informatiilor incapsulate in obiect,** (ansamblul valorilor curente ale atributelor obiectului) **trebuie specificat in mod explicit un nou cod (o noua implementare)** pentru metoda **`equals()`**. Acest lucru se obtine adaugand clasei din care face parte acel obiect o metoda cu declaratia:

```
public boolean equals(Object obj) { // urmeaza corpul metodei ...
```

metoda care **rescrie (*overrides*)** codul metodei cu acelasi nume din clasa extinsa (in acest caz clasa **`Object`**). Dupa adaugarea acestei metode, apelul **`equals()`** (sau **`this.equals()`**) va conduce la executia noului cod, pe cand apelul **`super.equals()`** va conduce la executia codului din clasa extinsa (in acest caz codul implicit din clasa **`Object`**).

Apelul constructorului din clasa extinsa se realizeaza utilizand apelul **`super()`,** apel care **trebuie sa fie prima instructiune din codul constructorului clasei care extinde**. In cazul nostru, constructorul clasei care extinde (**`PunctExtins`**) trebuie sa apeleze mai intai constructorul din clasa extinsa (**`Punct`**) utilizand **`super()`**, orice alt cod al constructorului **`PunctExtins()`** urmand dupa acest apel:

```
1    public PunctExtins(int a, int o, String id) {
2      super(a, o, id);
3      // orice alt cod de initializare ...
4    }
```

In aceste conditii, **o posibila rezolvare** este urmatoarea:

```
1  class PunctExtins extends Punct {
2    public PunctExtins(int a, int o, String id) {
3      super(a, o, id);
4    }
5    public boolean equals(Object obj) {
6      if ((obj != null) && (obj instanceof PunctExtins)) {
7        PunctExtins celalaltPunct = (PunctExtins)obj;
8        return ((this.x == celalaltPunct.x) &&
9                (this.y == celalaltPunct.y) &&
10               (this.numePunct.equals(celalaltPunct.numePunct)));
11     }
12     return false;
13    }
14 }
```

d) Sa se creeze diagrama UML de clase pentru codurile claselor `Punct` si `PunctExtins`.

Problema 2.3.

a) Sa se modifice implementarea constructorului `Punct()` al programul `Punct.java` pentru a realiza initializarea atributelor **`x`**, **`y`**, si **`numePunct`**, daca declaratia acestuia este urmatoarea:

```
public Punct(int x, int y, String numePunct)
```

Observatii:

1. Constructorul cu noua declaratie, daca ar utiliza urmatorul cod:

```
1      public Punct(int x, int y, String numePunct) {
2        x = x;                 // cod eronat din punct de vedere conceptual
3        y = y;                 // cod eronat din punct de vedere conceptual
4        numePunct = numePunct; // cod eronat din punct de vedere conceptual
5      }
```

nu ar initializa atributele `x`, **`y`** si **`numePunct`**, deoarece **`x`**, **`y`** si **`numePunct`** reprezinta parametrii constructorului. **In astfel de cazuri, se utilizeaza** cuvantul cheie **`this`**, care **contine referinta catre obiectul curent** (cel care e initializat de constructor), atributele fiind accesate sub forma **`this.x`**, **`this.y`** si respectiv **`this.numePunct`**.

2. O alta posibila greseala ar fi scrierea unui cod de genul:

```
1      public Punct(int x, int y, String numePunct) {
2        int x = x;             // cod eronat din punct de vedere conceptual
3        int y = y;             // cod eronat din punct de vedere conceptual
4        String numePunct = numePunct;       // cod eronat d.p.v. conceptual
5      }
```

care ar crea trei variabile locale (**al caror "scop"** – domeniu de existenta si vizibilitate in interiorul clasei - **este corpul constructorului** incepand de la declaratia variabilei locale si pana la sfarsitul codului constructorului) **care nu vor mai exista dupa ce constructorul isi incheie executia.** Astfel, **cele trei atribute ar ramane neinitializate.**

b) Sa se supraincarce numele constructorului `Punct()` al programul `Punct.java` cu o varianta fara parametri, care sa initializeze atributele **`x`**, **`y`**, si **`numePunct`**, cu valorile **`0`**, **`0`** si respectiv **`"O"`** (reprezentand originea sistemului de axe).

c) Sa se supraincarce numele constructorului `Punct()` al programul `Punct.java` cu o varianta cu declaratia:

```
public Punct(int xy, String numePunctDiagonala)
```

care sa initializeze atributele **`x`** si **`y`** astfel incat punctul rezultat sa se afle pe diagonala principala a sistemului de axe.

Problema 2.4.

Sa se adauge programului `Punct.java`:

- o metoda numita `setX()`, care sa primeasca un parametru **`a`** de tip **`int`**, sa nu returneze nici o valoare, si sa atribuie variabilei membru **`x`** valoarea parametrului primit,

- o metoda numita `setY()`,care sa primeasca un parametru **`o`** de tip **`int`**, sa nu returneze nici o valoare, si sa atribuie variabilei membru **`y`** valoarea parametrului primit.

- o metoda numita `setNume()`, care sa primeasca un parametru **`id`** de tip **`String`**, sa nu returneze nici o valoare, si sa atribuie variabilei membru **`numePunct`** valoarea parametrului primit.

- o metoda numita `getX()`, care sa nu primeasca nici un parametru, si sa returneze valoarea de tip **`int`** a atributului **`x`**.

- o metoda numita `getY()`, care sa nu primeasca nici un parametru, si sa returneze valoarea de tip **`int`** a atributului **`y`**.

- o metoda numita `getNume()`, care sa nu primeasca nici un parametru, si sa returneze valoarea de tip **`String`** a atributului **`numePunct`**.

Problema 2.5.

a) Care dintre urmatoarele linii de cod va produce eroare?

```
 1  public class UtilizarePunct {
 2    public static void main(String[] args) {
 3      Punct x = new Punct(3, 4, "X");
 4
 5      PunctExtins y = new Punct(5, 4, "Y");
 6
 7      Punct z = new PunctExtins(3, 2, "Z");
 8
 9      PunctExtins w = new PunctExtins(1, 4, "W");
10
11      Punct n = (Punct) new PunctExtins(3, -1, "N");
12
13      PunctExtins m = (PunctExtins) new Punct(5, -2, "M");
14    }
15  }
```

b) Care este tipul (clasa) fiecaruia dintre obiectele anterioare (`x, y, z, w, n, m`)?

Problema 2.6.

a) Ce iesire va produce pe ecran urmatorul de cod?

```
 1  public class Test1Equals {
 2    public static void main(String[] args) {
 3      Punct p1 = new Punct(2, 1, "P");  // obiect al clasei Punct
 4      Punct p2 = p1;                    // noua referinta spre acelasi obiect
 5      Punct p3 = new Punct(2, 1, "P");  // nou obiect cu acelasi continut
 6      Punct p4 = new Punct(3, 2, "S");  // nou obiect cu alt continut
 7
 8      if (p1 == p2) System.out.println("p1 == p2");
 9      else          System.out.println("p1 != p2");
10      if (p1 == p3) System.out.println("p1 == p3");
11      else          System.out.println("p1 != p3");
12      if (p1 == p4) System.out.println("p1 == p4");
13      else          System.out.println("p1 != p4");
14
15      if (p1.equals(p2)) System.out.println("p1.equals(p2)");
16      else               System.out.println("!p1.equals(p2)");
17      if (p1.equals(p3)) System.out.println("p1.equals(p3)");
18      else               System.out.println("!p1.equals(p3)");
19      if (p1.equals(p3)) System.out.println("p1.equals(p3)");
20      else               System.out.println("!p1.equals(p3)");)");
21    }
22  }
```

b) Ce iesire va produce pe ecran urmatorul de cod?

```
1  public class Test2Equals {
2    public static void main(String[] args) {
3      PunctExtins pex1 = new PunctExtins(2, 1, "P"); // obiect PunctExtins
4      PunctExtins pex2 = pex1;                       // noua referinta, acelasi obiect
5      PunctExtins pex3 = new PunctExtins(2, 1, "P"); // nou obiect, acelasi continut
6      PunctExtins pex4 = new PunctExtins(3, 2, "S");   // nou obiect, alt continut
7
8      if (pex1 == pex2) System.out.println("pex1 == pex2");
9      else              System.out.println("pex1 != pex2");
10     if (pex1 == pex3) System.out.println("pex1 == pex3");
11     else              System.out.println("pex1 != pex3");
12     if (pex1 == pex4) System.out.println("pex1 == pex4");
13     else              System.out.println("pex1 != pex4");
14
15     if (pex1.equals(pex2)) System.out.println("pex1.equals(pex2)");
16     else                   System.out.println("!pex1.equals(pex2)");
17     if (pex1.equals(pex3)) System.out.println("pex1.equals(pex3)");
18     else                   System.out.println("!pex1.equals(pex3)");
19     if (pex1.equals(pex4)) System.out.println("pex1.equals(pex4)");
20     else                   System.out.println("!pex1.equals(pex4)");
21   }
22 }
```

c) Ce iesire va produce pe ecran urmatorul de cod?

```
1  public class Test3Equals {
2    public static void main(String[] args) {
3      String s1 = new String("P");     // obiect al clasei String
4      String s2 = s1;                  // noua referinta spre acelasi obiect
5      String s3 = new String("P");     // nou obiect cu acelasi continut
6      String s4 = new String("S");     // nou obiect cu alt continut
7
8      if (s1 == s2) System.out.println("s1 == s2");
9      else          System.out.println("s1 != s2");
10     if (s1 == s3) System.out.println("s1 == s3");
11     else          System.out.println("s1 != s3");
12     if (s1 == s4) System.out.println("s1 == s4");
13     else          System.out.println("s1 != s4");
14
15     if (s1.equals(s2)) System.out.println("s1.equals(s2)");
16     else               System.out.println("!s1.equals(s2)");
17     if (s1.equals(s3)) System.out.println("s1.equals(s3)");
18     else               System.out.println("!s1.equals(s3)");
19     if (s1.equals(s4)) System.out.println("s1.equals(s4)");
20     else               System.out.println("!s1.equals(s4)");
21   }
22 }
```