2011 - 2012

Programare **O**rientata **spre O**biecte
(*Object-Oriented Programming*)

a.k.a. Programare Obiect-Orientata

Titular curs: Eduard-Cristian Popovici

Suport curs: *http://discipline.elcom.pub.ro/POO-Java/*

2. Orientarea spre obiecte in limbajul Java

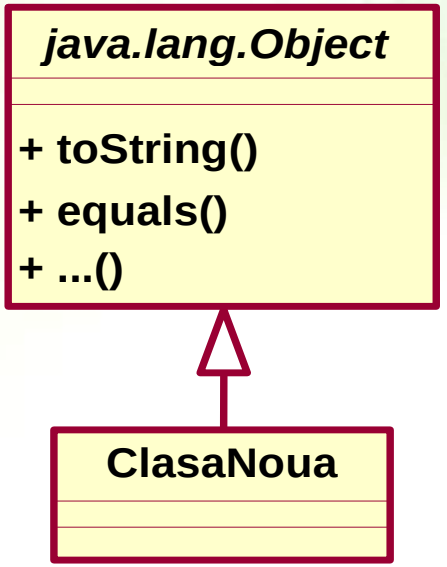
2.6. Polimorfismul metodelor

Mostenirea – mecanism suport pentru generalizare/specializare

Orice clasa Java care nu extinde prin mostenire **explicit o alta**

- **extinde implicit** clasa **Object** (radacina ierarhiei de clase Java)

  - care **contine metodele necesare tuturor obiectelor Java** (***equals()***, ***toString()***, etc.)



Declaratia:

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

este **echivalenta cu**:

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

Polimorfismul operatiilor (metodelor) in subclase

Printre metodele declarate in clasa **Object** este si **toString()**

- care are ca scop **returnarea sub forma de String** a **informatiilor pe care le incapsuleaza obiectul** (atributele) caruia i se aplica aceasta metoda

- In **cazul claselor de biblioteca** Java

  - metoda **toString()** returneaza **ansamblul valorilor curente ale atributelor** obiectului

- 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 este:

```
// Implementarea implicita a metodei toString(), mostenita de la clasa Object
public String toString() {

   // (nu returneaza continutul ci numele clasei si codul obiectului!)
   return getClass().getName() + "@" + Integer.toHexString(hashCode());
}
```

Polimorfismul operatiilor (metodelor) in subclase

➢ In **cazul in care programatorul doreste returnarea valorilor 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

```
// Implementarea explicita a metodei toString()
public String toString() {
   // urmeaza corpul metodei ...
}
```

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 Object**)

Polimorfismul operatiilor (metodelor) in subclase

Clasa Java care foloseste metoda **toString() mostenita de la clasa Object**

```
public class Punct {                                    // extinde Object!!!!!!!!!

   protected int x;                                     // atribute
   protected int y;
   protected String numePunct;

   public Punct(int a, int o, String id) {              // constructor
     x = a;
     y = o;
     numePunct = id;
   }

   public void afisarePunct() {                         // metoda non-statica
     System.out.println("Punctul " + this.toString()); // cod mostenit
   }

   public static void main(String[] args) {             // metoda principala
     Punct p = new Punct(2, 1, "P");
     p.afisarePunct();                                  // afiseaza   Punct@1add2dd
   }
}
```

Polimorfismul operatiilor (metodelor) in subclase

Daca **se modifica clasa adaugand o noua implementare** a **toString()**

```
public class Punct {                          // extinde Object!!!!!!!!
    protected int x;
    protected int y;
    protected String numePunct;

    public Punct(int a, int o, String id) {
      x = a;
      y = o;
      numePunct = id;
    }
    public void afisarePunct() {
      System.out.println("Punctul " + this.toString());  // cod nou
    }

    public String toString() {
      return this.numePunct + " (" + this.x + ", " + this.y + ")";
    }

    public static void main(String[] args) {
      Punct p = new Punct(2, 1, "P");
      p.afisarePunct();                          // afiseaza   "P (2, 1)"
    }
}
```

Polimorfismul operatiilor (metodelor) in subclase

Printre metodele declarate in clasa **Object** este si **equals()**

- care **compara**
  - **obiectul caruia i se aplica** aceasta metoda
  - cu un **obiect pasat ca parametru**

din punct de vedere al egalitatii starilor (adica **verifica egalitatea seturilor de valori curente ale atributelor**)

- **returnand** valoarea booleana **true** in **cazul egalitatii**
- si valoarea booleana **false** in **cazul inegalitatii** celor doua obiecte

Polimorfismul operatiilor (metodelor) in subclase

- In **cazul claselor de biblioteca** Java metoda **equals()** **compara obiectul** caruia i se aplica aceasta metoda **cu obiectul** pasat ca parametru

De exemplu, in **cazul clasei String:**

```
public boolean equals(Object obj) {       // obj este doar o referinta!!!
   // Verificare ca referinta primita refera un obiect, al clasei String
   if ((obj != null) && (obj instanceof String)) {
      String otherString = (String) obj;  // conversie sigura pentru ca
      int n = this.count;                 // instanceof a returnat true
      // Comparatie a numarului de caractere
      if (n == otherString.count) {
        char v1[] = this.value;       // pregatire a tablourilor de caractere
        char v2[] = otherString.value;
        int i = this.offset;          // pregatire a indecsilor in tablouri
        int j = otherString.offset;
        while (n-- != 0)              // pentru fiecare caracter din sir
          // Comparatie a valorilor caracterelor
          if (v1[i++] != v2[j++]) return false; // comparatia a esuat
        return true;                            // comparatia a reusit
      }
   }
   return false;                                // comparatia a esuat
}
```

Polimorfismul operatiilor (metodelor) in subclase

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** este:

```
// Implementarea implicita a metodei equals(), mostenita de la clasa Object

public boolean equals(Object obj) {

   // Nu compara continutul ci referintele!!!

   return (this == obj);

}
```

Polimorfismul operatiilor (metodelor) in subclase

In **cazul in care programatorul doreste comparatia obiectului caruia i se aplica** aceasta metoda **cu obiectul pasat ca parametru**

- **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

```
// Implementarea explicita a metodei equals()
public boolean equals(Object obj) {
   // urmeaza corpul metodei ...
}
```

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 Object**)

Polimorfismul operatiilor (metodelor) in subclase

Clasa Java care defineste metoda **equals()**

```
class PunctExtins extends Punct {

  public PunctExtins(int a, int o, String id) {
    super(a, o, id); // reutilizare constructor Point() prin apel cu super()
  }

  // cod nou (rescriere - override)
  public boolean equals(Object obj) {

    // Verificare ca referinta primita refera un obiect, tip PunctExtins
    // si ca atributul numePunct refera si el un obiect
    if ((obj != null) && (obj instanceof PunctExtins)
                      && (this.numePunct != null) ) {
       PunctExtins celalaltPunct = (PunctExtins) obj;

       // Returnare true daca toate atributele au valori egale, altfel false
       return ((this.x == celalaltPunct.x) &&
               (this.y == celalaltPunct.y) &&
               (this.numePunct.equals(celalaltPunct.numePunct)));
    }
    return false;     // comparatia a esuat
  }
}
```

Polimorfismul operatiilor (metodelor) in subclase

Care dintre urmatoarele linii de cod va **produce eroare?**

```
public class UtilizarePunct {

  public static void main(String[] args) {

    Punct x = new Punct(3, 4, "X");

    PunctExtins y = new Punct(5, 4, "Y");

    Punct z = new PunctExtins(3, 2, "Z");

    PunctExtins w = new PunctExtins(1, 4, "W");

    Punct n = (Punct) new PunctExtins(3, -1, "N");

    PunctExtins m = (PunctExtins) new Punct(5, -2, "M");
  }
}
```

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

Polimorfismul operatiilor (metodelor) in subclase

Care dintre urmatoarele linii de cod va **produce eroare?**

```
public class UtilizarePunct {

  public static void main(String[] args) {

    Punct x = new Punct(3, 4, "X");

    // eroare compilare  PunctExtins y = new Punct(5, 4, "Y");

    Punct z = new PunctExtins(3, 2, "Z");

    PunctExtins w = new PunctExtins(1, 4, "W");

    Punct n = (Punct) new PunctExtins(3, -1, "N");

    // eroare executie  PunctExtins m = (PunctExtins) new Punct(5,-2,"M");

  }
}
```

Care este tipul (clasa) fiecaruia dintre obiectele **x, y, z, w, n, m**? – cea **specificata de constructor**

Polimorfismul operatiilor (metodelor) in subclase

Ce iesire va produce pe ecran urmatorul de cod?

```
public class Test1Equals {
  public static void main(String[] args) {
    Punct p1 = new Punct(2, 1, "P");  // obiect al clasei Punct
    Punct p2 = p1;                    // noua referinta spre acelasi obiect
    Punct p3 = new Punct(2, 1, "P");  // nou obiect cu acelasi continut
    Punct p4 = new Punct(3, 2, "S");  // nou obiect cu alt continut

    if (p1 == p2) System.out.println("p1 == p2");
    else          System.out.println("p1 != p2");
    if (p1 == p3) System.out.println("p1 == p3");          // compara referinte
    else          System.out.println("p1 != p3");
    if (p1 == p4) System.out.println("p1 == p4");
    else          System.out.println("p1 != p4");

    if (p1.equals(p2)) System.out.println("p1.equals(p2)");
    else               System.out.println("!p1.equals(p2)");
    if (p1.equals(p3)) System.out.println("p1.equals(p3)"); //compara referinte
    else               System.out.println("!p1.equals(p3)");
    if (p1.equals(p3)) System.out.println("p1.equals(p3)");
    else               System.out.println("!p1.equals(p3)");;)");
  }
}
```

Polimorfismul operatiilor (metodelor) in subclase

Ce iesire va produce pe ecran urmatorul de cod?

```
public class Test2Equals {
  public static void main(String[] args) {
    PunctExtins pex1 = new PunctExtins(2, 1, "P"); // obiect PunctExtins
    PunctExtins pex2 = pex1;                       // noua referinta, acelasi obiect
    PunctExtins pex3 = new PunctExtins(2, 1, "P"); // nou obiect, acelasi continut
    PunctExtins pex4 = new PunctExtins(3, 2, "S"); // nou obiect, alt continut

    if (pex1 == pex2) System.out.println("pex1 == pex2");
    else              System.out.println("pex1 != pex2");
    if (pex1 == pex3) System.out.println("pex1 == pex3"); // compara referinte
    else              System.out.println("pex1 != pex3");
    if (pex1 == pex4) System.out.println("pex1 == pex4");
    else              System.out.println("pex1 != pex4");

    if (pex1.equals(pex2)) System.out.println("pex1.equals(pex2)");
    else                   System.out.println("!pex1.equals(pex2)");
    if (pex1.equals(pex3)) System.out.println("pex1.equals(pex3)");  // compara
    else                   System.out.println("!pex1.equals(pex3)"); // continut!!!
    if (pex1.equals(pex4)) System.out.println("pex1.equals(pex4)");
    else                   System.out.println("!pex1.equals(pex4)");
  }
}
```

Polimorfismul operatiilor (metodelor) in subclase

Ce iesire va produce pe ecran urmatorul de cod?

```
public class Test3Equals {
  public static void main(String[] args) {
    String s1 = new String("P");      // obiect al clasei String
    String s2 = s1;                   // noua referinta spre acelasi obiect
    String s3 = new String("P");      // nou obiect cu acelasi continut
    String s4 = new String("S");      // nou obiect cu alt continut

    if (s1 == s2) System.out.println("s1 == s2");
    else          System.out.println("s1 != s2");
    if (s1 == s3) System.out.println("s1 == s3");
    else          System.out.println("s1 != s3");
    if (s1 == s4) System.out.println("s1 == s4");
    else          System.out.println("s1 != s4");

    if (s1.equals(s2)) System.out.println("s1.equals(s2)");
    else               System.out.println("!s1.equals(s2)");
    if (s1.equals(s3)) System.out.println("s1.equals(s3)");
    else               System.out.println("!s1.equals(s3)");
    if (s1.equals(s4)) System.out.println("s1.equals(s4)");
    else               System.out.println("!s1.equals(s4)");
  }
}
```

Polimorfismul operatiilor (metodelor) in subclase

```
class A extends Object  {          // se putea si fara extends Object
  public void metoda() {
    System.out.println("metoda()");
  }
}

class B extends A {
  public void metoda(int x) {      // supraincarcarea numelui metodei metoda()
    System.out.println("metoda(int x)");
  }

  public void metoda(String y){    // supraincarcarea numelui metodei metoda()
    System.out.println("metoda(String y)");
  }
}

public class ExempleSupraincarcareNume {
  public static void main(String[] args){
    B var = new B();
    var.metoda();
    var.metoda(3);
    var.metoda("String");
  }
}
```

```
/* Iesire program:
   metoda()
   metoda(int x)
   metoda(String y) */
```

Polimorfismul operatiilor (metodelor) in subclase

```
class A extends Object { }  // clasa fara cod}
class B extends A {
  public void metoda() {     System.out.println("metoda in clasa B");  }
}
class C extends Object { }   // clasa fara cod

public class ExempleConversiiTip {
  public static void main(String[] args){
    Object var = new B();
    // Linia urmatoare NU ar putea fi compilata
    // var.metoda();
    // Linia urmatoare NU ar putea fi compilata
    // ((A) var).metoda();   // referinta de tip A actioneaza ca o filtrare
    // Linia urmatoare va fi compilata si executata
    ((B) var).metoda();
    // Linia urmatoare va fi compilata si executata
    B v1 = (B) var;         // nu merge fara (B)
    // Linia urmatoare NU ar putea fi executata
    // C v2 = (C) var;
    // Linia urmatoare NU ar putea fi compilata
    // C v3 = (B) var;
  }
}
```

```
/*
   Iesire program:
   metoda in clasa B
*/
```

Polimorfismul operatiilor (metodelor) in subclase

```
class A extends Object {
  public void metoda(){    System.out.println("metoda in clasa A");  }
}
class B extends A {
  public void metoda(){    System.out.println("metoda in clasa B");  }
}
public class ExempleRescriereCoduri {
  public static void main(String[] args){
    Object var = new B(); // Se instantiaza un obiect al clasei B

    // Linia urmatoare va fi compilata si executata
    ( (B) var).metoda();

    // Linia urmatoare va fi compilata si executata (comportament polimorfic)
    ( (A) var).metoda();

    // Linia urmatoare NU ar putea fi compilata
    // var.metoda();     // Referinta de tip Object actioneaza ca o filtrare

    var = new A(); // Se instantiaza un obiect al clasei A

    // Se invoca o metoda asupra ei
    ( (A) var).metoda();
  }
}
```

```
/* Iesire program:
   metoda in clasa B
   metoda in clasa B
   metoda in clasa A */
```

Polimorfismul operatiilor (metodelor) in subclase

```
class A extends Object { // clasa fara cod, mosteneste toString() din Object
}
class B extends A {      // rescrie toString() din Object
  public String toString(){    return "toString in clasa B";  }
}
class C extends B {      // rescrie toString() din B
  public String toString(){    return "toString in clasa C";  }
}

public class ExemplePolimorfism {
  public static void main(String[] args){
    Object varA = new A();
    String v1 = varA.toString();
    System.out.println(v1);

    Object varB = new B();
    String v2 = varB.toString();
    System.out.println(v2);

    Object varC = new C();
    String v3 = varC.toString();
    System.out.println(v3);
  }
}
```

```
/* Iesire program:
   A@111f71
   toString in clasa B
   toString in clasa C */
```