Java Basics

De la POO

Salt la: Navigare, căutare

Cuprins

[ascunde] [ascunde]

Tipuri primitive

Conform POO, orice este un obiect; totusi, din motive de performanta exista in Java niste tipuri de baza, care nu sunt clase. Aceste tipuri, numite tipuri **primitive**, corespund tipurilor de date cel mai des folosite si sunt prezentate aici, alaturi de spatiul ocupat si intervalul corespunzator de valori:

Tip primitiv	Dimensiune (biti)	Minim	Maxim
`boolean`	—	—	—
`char`	16	Unicode 0	Unicode $2^{16} - 1$
`byte`	8	-128	127
`short`	16	-2^{15}	$2^{15} - 1$
`int`	32	-2^{31}	$2^{31} - 1$
`long`	64	-2^{63}	$2^{63} - 1$
`float`	32	IEEE754	IEEE754
`double`	64	IEEE754	IEEE754
`void`	—	—	—

Se observa ca:

- Java **nu** poseda tipuri `unsigned`
- tipul `char` este pe 16 biti si intrebuinteaza **Unicode**.

Tipurile de date primitive sunt asemanatoare celor din C. Variabilele avand tipuri primitive sunt alocate pe stiva (exact ca in C), in contrast cu instantele claselor, care sunt alocate pe heap.

Exemplu de declaratie:

```
int i, j;
boolean k;
```

Celelalte tipuri existente sunt clase. Instantele claselor sunt tipuri **referinta**. Acest lucru inseamna ca in spate masina virtuala Java lucreaza cu pointeri la obiecte si noi folosim pointeri impliciti catre zone de memorie alocate pentru obiectele utilizate. Acest lucru va avea consecinte mai tarziu, cand vom studia transferul parametrilor in Java.

Acum e de retinut faptul ca in Java **nu** exista pointeri expliciti. Aceasta facilitate a fost considerata generatoare de erori si nu a fost implementata. Acest lucu nu limiteaza capacitatile platformei.

Clase

Clasele reprezinta tipuri de date definite de utilizator sau deja existente in sistem (din *class library*-ul mentionat anterior). O clasa poate contine:

- **campuri** (*variabile membru*, care definesc starea obiectului)
- **metode** (*functii membru*, ce reprezinta operatii asupra starii).

Prin instantierea unei clase se intelege crearea unui obiect care corespunde tiparului definit de clasa respectiva. In cazul general, acest lucru se realizeaza prin intermediul cuvantului cheie `new`.

Biblioteca Java ofera clase **wrapper** ("ambalaj") pentru fiecare tip primitiv. Avem astfel clasele `Char`, `Integer`, `Float` etc. Un exemplu de instantiere este urmatorul:

```
Integer zero = new Integer(0);
```

Un alt exemplu de clasa predefinita este clasa `String`. Ea se instantiaza astfel (**nu** este necesara utilizarea `new`):

```
String s1, s2;
s1 = "Primul meu string";
s2 = "Al doilea string";
```

Aceasta este varianta preferata pentru instantierea string-urilor. De remarcat ca si varianta

```
s = new String("str");
```

este corecta, dar **ineficienta**, din motive ce vor fi explicate ulterior.

Functii membru

Putem modifica programul anterior astfel:

```
String s1, s2;
s1 = "Primul meu string";
s2 = "Al doilea string";
System.out.println(s1.length());
System.out.println(s2.length());
```

Va fi afisata lungimea in caractere a sirului respectiv. Se observa ca pentru a aplica o functie a unui obiect, se foloseste sintaxa:

```
instantaClasei.numeDeFunctie(param1, param2,..., paramn);
```

Functiile membru se declara asemanator cu functiile din C.

Variabile membru

Un camp este un obiect avand tipul unei clase sau o variabila de tip primitiv. Daca este un obiect atunci trebuie initializat inainte de a fi folosit (folosind cuvantul cheie `new`).

```
class DataOnly {
    int i;
    float f;
    boolean b;
    String s;
}
```

Declaram un obiect de tip `DataOnly` si il initializam:

```
DataOnly d = new DataOnly();
// setam campul i al obiectului d la valoarea 1
d.i = 1;
// folosim valoarea setata
System.out.println("Campul i al obiectului d este " + d.i);
```

Observam ca pentru a utiliza un camp/functie membru dintr-o functie care nu apartine clasei respective, folosim **sintaxa**:

```
instantaClasei.numeCamp
```

Pentru a utiliza un camp dintr-o functie a clasei respective, folosim **direct** numele campului, ca in exemplul urmator.

Clasa `VeterinaryReport` este o versiune micsorata a clasei care permite unui veterinar sa tina evidenta animalelor tratate, pe

categorii (caini/pisici):

```
public class VeterinaryReport {
    int dogs;
    int cats;

    public int getAnimalsNo() {
        return dogs + cats; // utilizare camp din acelasi obiect pentru care s-a facut apelul functiei
    }

    public void displayStatistics() {
        System.out.println("Total number of animals is " + getAnimalsNo()); // apel metoda din aceeasi clasa
   }
}
```

Clasa `VeterinaryTest` ne permite sa testam functionalitatea oferita de clasa anterioara.

```
public class VeterinaryTest {
    public static void main(String[] args) {
        VeterinaryReport vr = new VeterinaryReport(); // se creeaza un obiect de tipul VeterinaryReport
        vr.cats = 99; // se seteaza valori pentru variabilele membre ale obiectului vr (obiectul este specificat
exlpicit!)
        vr.dogs = 199;
        vr.displayStatistics(); // se apeleaza functia membra a obiectului creat
        System.out.println("The class method sais there are " + vr.getAnimalsNo() + " animals");
    }
}
```

Observatii:

- Daca **nu** initializam valorile campurilor explicit, masina virtuala va seta toate referintele la `null` si tipurile primitive la 0 (pentru tipul `boolean` la `false`).
- In Java **fisierul** trebuie sa aiba numele clasei (publice) care e continuta in el. Cel mai simplu si mai facil din punctul de vedere al organizarii codului este de a avea fiecare clasa in propriul fisier. In cazul nostru, `VeterinaryReport.java` si `VeterinaryTest.java`.
- O observatie importanta este ca obiectele au fiecare **propriul** spatiu de date: fiecare camp are o valoare separata pentru fiecare obiect creat. Codul urmator arata aceasta situatie:

```
public class VeterinaryTest {
    public static void main(String[] args) {
        VeterinaryReport vr = new VeterinaryReport();
        VeterinaryReport vr2 = new VeterinaryReport();
        vr.cats = 99;
        vr.dogs = 199;
        vr2.dogs = 2;
        vr.displayStatistics();
        System.out.println("The class method sais there are " + vr.getAnimalsNo() + " animals");
    }
}
```

Se observa ca, desi campul `dogs` apartinand obiectului `vr2` a fost updatat la valoarea 2, campul `dogs` al obiectului `vr1` a ramas la valoarea initiala (199). Fiecare obiect are spatiul lui pentru date!

Cuvantul cheie `this`

Dupa cum am mai spus, apelurile de functii membru din interiorul unei functii apartinand aceleiasi clase, se face direct prin nume. Apelul de functii apartinand unui alt obiect se face prefixand apelul de functie cu numele obiectului (exemplul `VeterinaryTest`/`VeterinaryReport`). Situatia este aceeasi pentru datele membru.

Totusi, unele functii pot trimite un parametru cu **acelasi** nume ca si un camp membru. In aceste situatii, cuvantul cheie `this` se foloseste pentru **dezambiguizare**, el prefixand denumirea campului cand se doreste utilizarea acestuia. Acest lucru este necesar pentru ca in Java este comportament default ca un nume de parametru sa ascunda numele unui camp.

Adaugam clasei `VeterinaryTest` o metoda care permite setarea numarului de animale tratate:

```
public class VeterinaryReport {
    int dogs;
    int cats;
    [...]
    public void setAnimalsNo(int cats, int dogs) {
        this.dogs = dogs;
        this.cats = cats;
    }
}
```

Vom trata cuvantul cheie `this` intr-un laborator urmator mai in detaliu.

Exercitii

1. Urmatorul tutorial video arata cum se pot crea proiecte, clase, metode noi exploatand facilitatile oferite de Eclipse.
2. (**1p**) Numere complexe
   - Creati un proiect nou cu numele `NumereComplexe`.
   - Creati clasa `NumarComplex.java`. Aceasta va avea doua campuri: `re` si `im`, ambele de tip `float`.
3. (**1p**) Operatii
   - Creati clasa `Operatii.java`. Aceasta clasa va implementa operatiile de adunare si inmultire pentru numere complexe. Definiti in clasa `Operatii` cate o metoda pentru fiecare operatie;
4. (**0.5p**) Testati metodele implementate.
5. (**3p**) Biblioteca
   - Creati un proiect nou cu numele `Biblioteca`.
   - Creati clasa `Carte.java` cu urmatoarele atribute: `titlu`, `autor`, `editura`, `numarPagini`.
   - Creati clasa `Test.java`, pentru a testa viitoarele functionalitati ale bibliotecii. Completati aceasta clasa, asa cum considerati necesar. Apoi, creati un obiect de tip carte si setati atributele introducand date de la tastatura. Pentru aceasta folositi clasa `Scanner`:
     - ```
       Scanner s = new Scanner(System.in);
       String titlu = s.nextLine();
       ```
     - Verificati ca numarul de pagini introdus sa fie diferit de zero. Puteti consulta documentatia claselor `String` si `Integer`.
6. (**3p**) Verificari carti
   - Creati o clasa noua, `Verificari`, ce va contine doua metode, cu cate 2 parametri de tipul `Carte`.
     - Prima metoda va verifica daca o carte este in dublu exemplar, caz in care va intoarce adevarat
     - A doua metoda va verifica care carte este mai groasa decat alta, si va intoarce cartea mai groasa.
   - Testati aceste doua metode.
7. (**2p**) Formatare afisare
   - Afisati informatia despre o carte in felul urmator: TITLUL_CARTII, Numele_Autorului, numele_editurii (titlul cartii va fi scris in intregime cu **majuscule**, iar numele editurii va fi scris in intregime cu **minuscule**, numele autorului ramanand **neschimbat**). Puteti consulta documentatia clasei `String`.

Solutii

Adus de la "http://cursuri.cs.pub.ro/~poo/wiki/index.php/Java_Basics"
       ```

- Versiune de tipărit
- Legătură permanentă
- Print as PDF

- Ultima modificare 14:15, 4 noiembrie 2012.
- Această pagină a fost vizitată de 8.622 ori.
- Conţinutul este disponibil sub GNU Free Documentation License 1.2.
- Politica de confidenţialitate
- Despre POO
- Termeni