Organizarea surselor si controlul accesului

De la POO

Salt la: Navigare, căutare

Pachete

Chiar si in cadrul proiectelor de dimensiune medie, numarul claselor definite poate creste considerabil. Astfel, devine aparenta necesitatea unei organizari a fisierelor sursa, pe baza functiei indeplinite si relatiilor dintre acestea. In plus, aceasta organizare permite si utilizarea unor mecanisme de control al accesului.

Spre exemplu, in cadrul unei aplicatii de transfer de fisiere, se pot distinge mai multe module, precum cel de interfata cu utilizatorul, cel de retea, cel de configurare etc. In acest caz, este utila gruparea claselor aferente fiecarui modul in propriul pachet. Acestea se reflecta in directoare din sistemul de fisiere, putand imprumuta structura ierarhica a acestora. Astfel, definitia clasei `Main` din pachetul principal `myapp` ar incepe prin cuvantul cheie `package`.

```
package myapp;

import java.util.*;

public class Main {
    ...
}
```

Presupunand ca directorul ce contine sursele este `src`, fisierul `Main.java` se va gasi in directorul `src/myapp`.

In continuare, pentru implementarea modulului de retea, se poate defini pachetul `network`, care, in mod natural, va fi ierarhic inferior pachetului `myapp`. In astfel de situatii numele de pachete se separa prin " . ", ca in exemplul urmator:

```
package myapp.network;

public class NetworkConnection {
    ...
}
```

Fisierul `NetworkConnection.java` se va gasi in directorul `src/myapp/network`.

In functie de pozitionarea relativa a doua clase (in acelasi pachet sau in pachete diferite), fiecare poate accesa diferit membrii celeilalte, dupa cum se discuta in sectiunea urmatoare.

O documentatie detaliata este disponibila aici.

Specificatori de acces

Clasele si functiile mentionate pana acum au fost declarate utilizand un specificator special: `public`. In limbajul Java (si in majoritatea limbajelor de programare de tipul OOP), orice clasa, atribut sau metoda poseda un specificator de acces, al carui rol este de a restrictiona accesul la entitatea respectiva, din perspectiva altor clase. Exista specificatorii:

- `public` - permite acces complet din exteriorul clasei curente
- `private` - limiteaza accesul doar in cadrul clasei curente
- `protected` - limiteaza accesul doar in cadrul clasei curente si al tuturor descendentilor ei (acest concept va fi explicat mai tarziu)
- `default` - in cazul in care nu este utilizat explicit nici unul din specificatorii de acces de mai sus, accesul este permis doar in cadrul pachetului (*package private*). Atentie, nu confundati specificatorul *default* cu *protected*.

Utilizarea specificatorilor contribuie la realizarea **incapsularii**. Amintim, din primul laborator, ca incapsularea se refera la acumularea atributelor si metodelor caracteristice unei anumite categorii de obiecte intr-o clasa. Pe de alta parte, acest concept denota si *ascunderea* informatiei de stare a unui obiect, reprezentata de atributele acestuia, alaturi de valorile aferente, si asigurarea comunicarii strict prin intermediul metodelor (*interfata* clasei). Acest lucru conduce la **izolarea** modului de implementare a unei clase (atributele acesteia si felul in care metodele le manipuleaza) de utilizarea acesteia. Utilizatorii unei clase pot conta pe functionalitatea expusa de aceasta, indiferent de modalitatea in care ea este implementata, aceasta putandu-se chiar modifica in timp. Accesul utilizatorilor la implementarea unei clase ar conduce la imposibilitatea modificarii acesteia din urma, fara a declansa actualizari ale portiunilor ce utilizeaza clasa respectiva.

Exercitii

Nota: Exercitiile se rezolva in ordine.

1. (**5p**) Sa se implementeze o clasa `MyArrayList`, care va reprezenta un vector de numere reale, cu posibilitatea redimensionarii

automate. Ea va fi definita in pachetul `arrays`. Clasa va contine urmatoarele metode:
   - un constructor fara parametri, care creeaza intern un vector cu 10 elemente
   - un constructor cu un parametru de tipul intreg, care creaza un vector de dimensiune egala cu parametrul primit
   - o metoda numita `add(float value)`, care adauga valoarea `value` la finalul vectorului. Daca se depaseste capacitatea vectorului, acesta se va redimensiona la o capacitate dubla. **Atentie!** Exista o **diferenta** intre capacitatea vectorului (dimensiunea cu care a fost initializat) si numarul de elemente memorate efectiv in el (care este cel mult capacitatea).
   - o metoda numita `contains(float value)` care returneaza `true` daca `value` exista in cadrul vectorului
   - o metoda numita `remove(int index)` care elimina valoarea aflata in vector la pozitia specificata de `index` (numerotarea incepand de la 0). Se va da un mesaj daca indexul este invalid.
   - o metoda numita `get(int index)` care va returna elementul aflat in pozitia `index`
   - o metoda numita `size()` care returneaza numarul de elemente din vector.
   - o metoda declarata `public String toString()` care va returna o reprezentare a tuturor valorilor vectorului intr-un sir de caractere.
2. (**1p**) Scrieti o clasa de test **separata** pentru clasa `MyArrayList`, populand lista cu 3 elemente si verificand ca valorile intoarse de metoda `get` corespund, intr-adevar, pozitiilor aferente din vectorul intern clasei. Ce conditii trebuie indeplinite pentru atingerea scopului propus?
3. (**2p**) Scrieti un scenariu de utilizare a clasei `MyArrayList`, astfel:
   - initializand-o cu o capacitate de 5 de elemente iar apoi inserand 10 elemente aleatoare utilizand doar metoda `add`
   - cautand 5 valori aleatoare in vector
   - eliminand 5 valori aleatoare din vector.

   Dupa fiecare din acesti 3 pasi, afisati continutul vectorului utilizand metoda `toString()`. Pentru a genera valori aleatoare, utilizati:
   ```
   import java.util.*;
   Random generator = new Random();
   float value = generator.nextFloat();
   ```
4. (**3p**) De multe ori este bine ca programarea unor aplicatii mari sa se faca modular, un modul fiind gestionat de o anumita clasa. De asemenea, in cadrul acestor aplicatii, puteti avea situatii in care o metoda are nevoie sa utilizeze mai multe module, deci sa primeasca referintele mai multor clase, fapt ce poate deveni dificil de gestionat. De aceea, aceste clase se pot implementa limitand accesul la o singura instanta statica, ceea ce permite accesarea acesteia doar pe baza clasei, fara a mai fi necesara trimiterea ei ca parametru in metode. Acest mecanism mai poarta numele de **`Singleton pattern`**.
   Pornind de la aceasta idee, implementati o clasa `Singleton`, utilizand in cadrul acesteia:
   - o **instanta statica** a clasei
   - un **constructor privat**
   - o **metoda publica** de acces la instanta (un *getter*)

   Justificati utilizarea unui constructor privat.

- **Solutii**

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