Programare orientată pe obiecte

#15 JAVA
Pachete. Interfețe

https://www.runceanu.ro/adrian/activitate-didactica/

Curs 15

Pachete. Interfețe

Pachete. Interfețe

1. Pachete de clase de obiecte

2. Moştenire multiplă. Interfeţe Java

Ce este un pachet?

- Un pachet (package) contine un numar de clase inrudite ca scop, ca domeniu sau din punct de vedere al mostenirii.
- Daca programele sunt mici si folosesc un numar limitat de clase, nu este necesar sa cream pachete.
- Insa daca aplicatia noastra foloseste din ce in ce mai multe clase atunci este necesar sa le organizam in pachete.

De ce sunt folositoare pachetele?

- *Pachetele permit organizarea claselor in grupuri*

- Asa cum pe hard-disc avem directoare si subdirectoare pentru a ne organiza fisierele si aplicatiile, pachetele ne permit sa organizam clasele in grupuri din care putem folosi doar ceea ce avem nevoie pentru fiecare aplicatie.

Proprietati ale pachetelor

- *Pachetele reduc problemele datorate conflictelor de nume*.
- Cu cat creste numarul claselor de obiecte, cu atat creste posibilitatea de a folosi un nume de clasa deja existent, ceea ce va duce la aparitia unor conflicte si erori la integrarea acestora in aplicatii.
- Pachetele permit "*ascunderea claselor*" si evitarea acestor conflicte.

Proprietati ale pachetelor

- *Pachetele protejeaza clasele, variabilele si metodele depasind nivelul de clasa*.

- Pachetele pot fi folosite la identificarea claselor.

- De exemplu, daca implementam un set de clase pentru a realiza o anumita sarcina, putem sa folosim pentru pachetul de clase respectiv un identificator unic, care sa desemneze autorul sau organizatia din care provine.

Proprietati ale pachetelor

Nota:

- Chiar daca un pachet este, in esenta, un grup de clase, acesta poate contine si alte pachete, care formeaza alt nivel ierarhic asemanator, cumva, ierarhiei de clase.

- Fiecare nivel reprezinta, de obicei, o grupare mai mica de clase si cu sarcini mai precise.

- Biblioteca de clase Java este organizata si ea dupa aceste principii.

Utilizarea pachetelor existente de clase de obiecte

- Stim ca pachetele grupeaza mai multe clase la un loc.

- Pana acum am folosit pachete din biblioteca Java in legatura cu operatiile de I/O necesare citirii de la tastatura a datelor, pentru generarea unor obiecte de tip *Random,* etc.

Utilizarea pachetelor existente de clase de obiecte

Ca regula, putem alege unul dintre urmatoarele trei mecanisme pentru a utiliza o clasa continuta intr-un pachet:

1. Atunci cand clasa pe care dorim sa o folosim se afla in pachetul java.lang, ea se poate referi pur si simplu prin numele ei

2. Atunci cand clasa pe care dorim sa o folosim se afla in alt pachet, ea se poate referi folosindu-i numele complet, adica inclusiv pe cel al pachetului (*java.util.Random*)

Utilizarea pachetelor existente de clase de obiecte

3. Pentru clasele din alte pachete folosite frecvent in aplicatie, se pot importa clasele individuale sau intreg pachetul de clase folosit; dupa ce clasa sau pachetul au fost importate, se poate referi clasa doar prin numele sau.

Nota:

- Clasele care nu sunt declarate ca facand parte dintr-un anumit pachet sunt automat incluse intr-un pachet prestabilit.
- Aceste clase pot fi referite prin numele lor, de oriunde din cod.

Comanda import

- Pentru a importa clasele dintr-un pachet se foloseste comanda ***import***, asa cum am vazut in multe exemple prezentate pana acum.

- Se poate importa o clasa ca in exemplul urmator:

import java.util.Vector;

- Se poate importa un intreg pachet de clase, folosind simbolul asterisc (*) in locul numelor de clasa, astfel:

import java.io.*;

Comanda import

Nota:

- Comanda din exemplul de mai sus nu importa toate clasele pachetului; sunt importate doar acele clase care au fost declarate *public*, si chiar si asa sunt importate doar acele clase care sunt referite in cod.

- De asemenea, daca pachetul respectiv are si subpachete, prin comanda ***import*** din exemplul de mai sus, nu se vor importa si subpachetele pachetului specificat in comanda ***import***.

Comanda import

- Pentru a importa toate clasele dintr-o ierarhie complexa de pachete va trebui sa se importe explicit fiecare nivel al ierarhiei.
- Instructiunea ***import*** trebuie plasata inaintea oricarei definitii de clasa (insa dupa definitia pachetului, daca se doreste crearea unui pachet propriu).
- Spre deosebire de directiva *#include* din limbajul C/C++, care include cod-sursa dintr-un alt fisier, instructiunea ***import*** indica doar locul unde poate fi gasita o clasa si nu are nici un rol in marirea dimensiunii aplicatiei proiectate.

Comanda import

Observatie:

- Trebuie evitate *conflictele de nume de clasa*, deoarece compilatorul va semnala o eroare.
- *Conflicul de nume de clasa are loc atunci cand doua clase din pachete diferite au acelasi nume*.
- In aceasta situatie, referirea la clasa dorita din aceste pachete trebuie sa se faca prin numele sau complet (inclusiv numele pachetului).

Variabila CLASSPATH si locul de dispunere a claselor in sistemul de fisiere

- Pentru ca programul nostru sa poata folosi o clasa, trebuie sa afle locul unde este dispusa clasa in sistemul de fisiere; altfel, va fi generata o eroare referitoare la inexistenta clasei.

Pentru gasirea claselor, Java foloseste:

1. numele pachetului

2. directoarele referite de variabile de mediu CLASSPATH

1. Numele pachetelor corespund unor nume de directoare ale sistemului de fisiere.

- De exemplu, clasa *java.applet.Applet* se va gasi in directorul *applet* care face parte si el din directorul *java* (adica, java\applet\Applet.class).

2. Java cauta aceste directoare, pe rand, in cadrul cailor (path-urilor) incluse in variabila de mediu CLASSPATH, daca aceasta este definita.

- Daca nu este configurata nici o variabila CLASSPATH, Java cauta in directorul prestabilit, *java\lib* (aflat in directorul cu versiunea de soft SDK folosita), precum si in directorul curent.

- Java cauta, in aceste directoare, clasa referita in fisierul-sursa folosindu-se de numele pachetului si al clasei, iar daca nu o gaseste semnaleaza eroare de tip "*class not found*".

Crearea propriilor pachete de clase de obiecte

- Pentru crearea unor pachete proprii de clase de obiecte se parcurg urmatoarele etape:

1. Alegerea unui nume pentru pachet.

- Numele ales pentru pachet depinde de modul in care dorim sa folosim aceste clase.
- Conventia de denumire a pachetelor, recomandata de **Sun** este de a folosi *numele de domeniu Internet cu elementele inversate*.
- Ideea este de a ne asigura ca numele pachetului creat este unic.

Crearea propriilor pachete de clase de obiecte

- *Prin conventie, numele pachetelor incep cu litera mica, pentru a le deosebi de numele claselor, care incep cu litera mare*.
- De exemplu, in cazul denumirii complete a clasei *String* (adica, *java.lang.String*) putem vizualiza foarte usor numele pachetului de numele clasei.
- Aceasta conventie ajuta si ea la reducerea conflictelor de nume.

Crearea propriilor pachete de clase de obiecte

2. Crearea structurii de directoare.

- *Se creeaza pe hard-disc o structura de directoare conforma cu numele pachetelor*.
- Daca pachetul are un singur nume (cum ar fi: *pachetulmeu*) este suficient sa cream un director cu acest nume.
- Daca numele pachetului este compus din mai multe parti (cum ar fi: *ro.ucb.pachetulmeu*), trebuie create si subdirectoarele respective.
- De exemplu, pentru pachetul *ro.ucb.pachetulmeu* trebuie sa cream directorul *ro*, in cadrul acestuia subdirectorul *ucb*, iar in cel din urma, subdirectorul *pachetulmeu*.
- Clasele si fisierele sursa vor fi pastrate apoi in directorul *pachetulmeu*.

Crearea propriilor pachete de clase de obiecte

3. Adaugarea unei clase intr-un pachet.

- Pasul final este de a introduce clasa intr-un pachet, folosind instructiunea ***package***, plasata inainte de orice instructiune *import* folosita.
- Se poate adauga o clasa la un pachet ca in exemplul urmator:

```
package ro.ucb.pachetulmeu;
```

Nota:

- Folosirea unui mediu de dezvoltare de aplicatii de tipul **Eclipse** usureaza crearea pachetelor si a structurii de directoare aferente acestora.
- Mediile de acest tip permit si crearea de proiecte care inglobeaza pachetele si alte fisiere necesare aplicatiei, creand de fapt aplicatii de tip *.jar*.

Pachetele de clase si controlul accesului la clase

- Controlul accesului la clase se poate realiza folosind modificatorul de acces *public*.
- Acest acces inseamna ca respectiva clasa este vizibila si poate fi importata in afara pachetului.
- *Clasele declarate publice pot fi importate de orice alte clase din afara pachetului.*

Pachetele de clase si controlul accesului la clase

- Atunci cand folosim instructiunea *import* cu simbolul * , se vor importa numai clasele publice din pachetul respectiv.
- Daca nu este specificat nici un modificator pentru clasa, atunci acesteia i se atribuie un control prestabilit al accesului de tip *package-friendly*.
- Acest acces presupune ca respectiva clasa este disponibila (vizibila) tuturor claselor din acelasi pachet, insa nu si in afara pachetului - nici macar subpachetelor.
- Clasa nu poate fi importata sau referita prin nume.

Exemplu de pachete proprii de clase create pentru aplicatia TestFormaInterfata.java

Ne propunem sa rezolvam urmatoarea problema:

Se citesc N forme geometrice (patrate, dreptunghiuri, cercuri) de la tastatura.

Sa se afiseze formele geometrice ordonate dupa arie.

Exemplu de pachete proprii de clase create pentru aplicatia TestFormaInterfata.java

Pentru rezolvarea acestei probleme s-au definit urmatoarele clase de obiecte:

- clasa abstracta a formelor geometrice cu numele *FormaGeo*;

- clasele derivate din superclasa *FormaGeo* care caracterizeaza formele geometrice specifice: *Cerc*, *Dreptunghi* si *Patrat*;

Exemplu de pachete proprii de clase create pentru aplicatia TestFormaInterfata.java

- doua clase ajutatoare, care pot fi folosite si de alte aplicatii si anume:

  – interfata *Compara* care declara o metoda publica cu numele *comparaCu()*, asemanatoare metodei din biblioteca Java *compareTo()*, folosita pentru compararea ariei a doua obiecte de tip *FormaGeo*;

  – clasa *Sort* care realizeaza ordonarea formelor geometrice concrete (Cerc, Patrat si Dreptunghi) dupa arie; acesta clasa are un caracter mai general si poate fi folosita pentru ordonarea altor obiecte dupa diferite criterii.

Exemplu de pachete proprii de clase create pentru aplicatia TestFormaInterfata.java

- clasa aplicatiei propriu-zise cu numele *TestFormaInterfata* folosita pentru citirea formelor de la tastatura si afisarea formelor geometrice ordonate dupa arie.

- Pentru o mai buna organizare, clasele de obiecte enumerate au fost impartite pe pachete, ceea ce inseamna ca fiecare fisier de tip *.class* a fost stocat intr-un director corespunzator numelui pachetului.

Exemplu de pachete proprii de clase create pentru aplicatia TestFormaInterfata.java

- Astfel, au fost create urmatoarele pachete de clase de obiecte:
  - pachetul cu numele *clasegenerice* care contine fisierele de tip *.class*: *Sort.class* si *Compara.class*; acest pachet a fost stocat in directorul cu numele *clasegenerice*;
  - pachetul cu numele *formegeometrice* care contine fisierele de tip *.class*: *Cerc.class*, *Dreptunghi.class*, *Patrat.class* si *FormaGeo.class*; acest pachet a fost stocat in directorul cu numele *formegeometrice*.

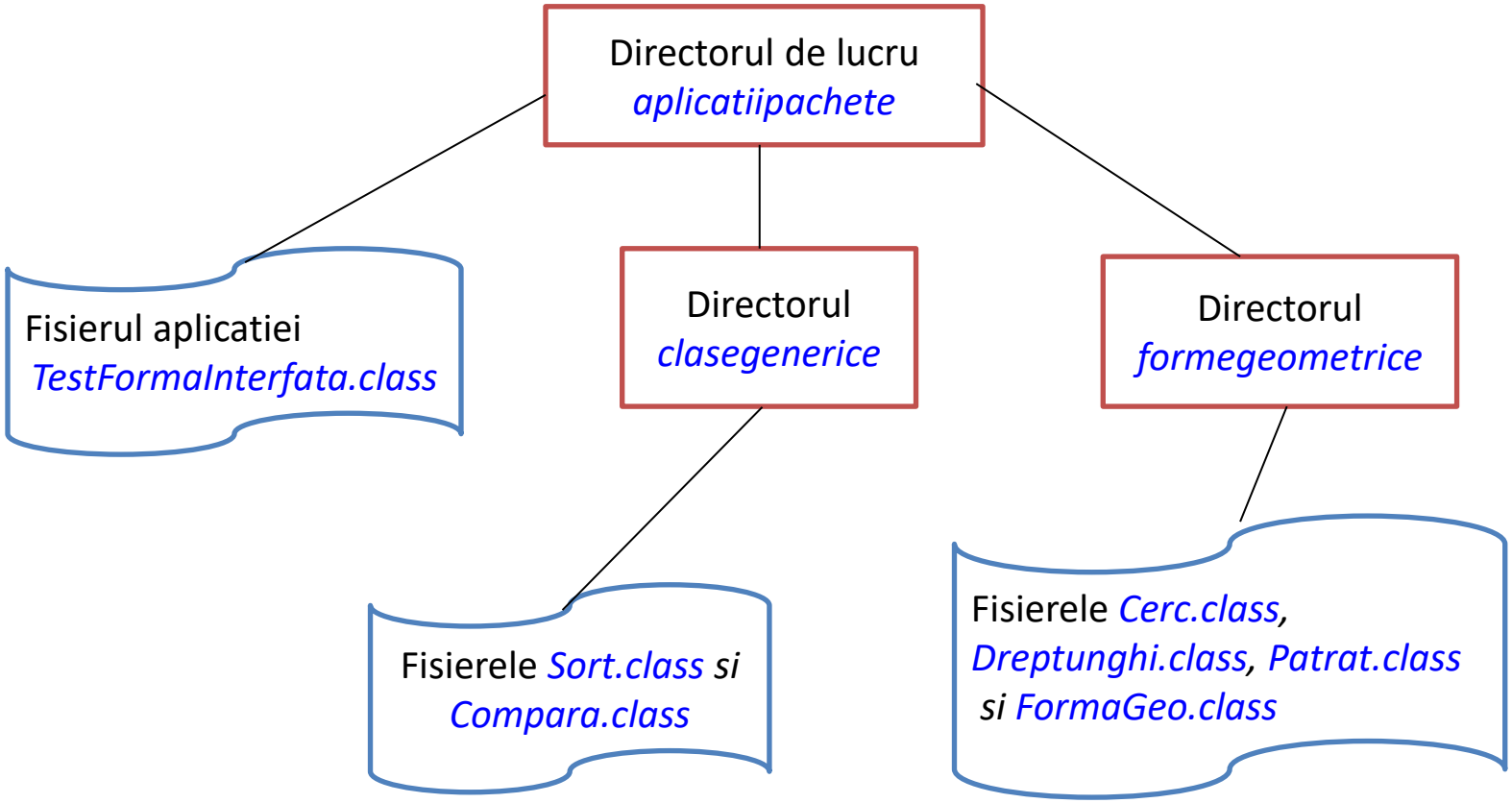
Exemplu de pachete proprii de clase create pentru aplicatia TestFormaInterfata.java

- Fisierul *TestFormaInterfata.class* corespunzator aplicatiei propriu-zise nu a fost inclus in nici un pachet si a fost stocat in directorul curent de lucru *aplicatiipachete*.

- Pentru ca aplicatia sa functioneze, acest director trebuie adaugat la variabila de mediu CLASSPATH.

- In directorul curent s-au creat si directoarele corespunzatoare pachetelor descrise.

In concluzie, structura de directoare si pachetele continute in aceste directoare arata ca in schema de mai jos:



Pachete. Interfete

1. Pachete de clase de obiecte

2. Moştenire multiplă. Interfeţe Java

Mostenire multipla

- In cazul ***mostenirii multiple*** *o clasa este derivata din doua sau mai multe superclase*.

- De exemplu, pot exista in ierarhie clase precum *Student* si *Angajat*.

- Din aceste doua clase ar putea fi derivata o clasa cu numele *AngajatStudent* care sa contina atribute si metode combinate din clasele *Student* si *Angajat*.

Mostenire multipla

- *Mostenirea multipla poate conduce insa la dificultati de proiectare.*

- De exemplu, cele doua superclase din care deriveaza subclasa ar putea contine metode care au aceeasi semnatura, dar implementari diferite sau ar putea avea atribute cu acelasi nume.

- Dificultatea rezolvarii unor astfel de probleme a facut ca Java sa nu permita mostenirea multipla.

Interfete Java

- Alternativa oferita de Java pentru mostenirea multipla este ***folosirea interfetelor***.

- O interfata Java este *o colectie de comportamente abstracte, care pot fi combinate in orice clasa pentru a introduce in acea clasa comportamente care nu pot fi mostenite de la superclasa*.

- Tehnic, **interfata Java** este cea mai abstracta clasa posibila.
- Ea consta doar din *metode publice abstracte* si din *atribute statice si finale*.

Interfete Java

- *O clasa implementeaza o anumita interfata* daca furnizeaza definitii pentru toate metodele abstracte din cadrul interfetei.

- O clasa care implementeaza o interfata se comporta ca si cand ar fi extins o clasa abstracta specificata de acea interfata.

Crearea si folosirea interfetelor Java

Sintaxa de definire a unei interfete este urmatoarea:

[<modificatori>] **interface** <nume_interfata>
  [**extends** <nume_interfata1>
  [, <nume_interfata2>]
  [,nume_interfata3>], …]]
{
  <corpul interfetei>
}

unde:

- <modificatori> - sunt specificati prin cuvintele-cheie *public* si *abstract*; o interfata publica poate fi accesata si de alte pachete decat cel care a definit-o; fiecare interfata este in mod implicit abstracta deci cuvantul-cheie *abstract* poate lipsi;
- <nume_interfata> - specifica numele interfetei; este de preferat ca numele interfetei sa respecte aceeasi conventie de numire ca si cea de numire a claselor;
- <nume_interfata1>, <nume_interfata2>, ... - specifica numele superinterfetelor din care poate deriva interfata;
- <corpul_clasei> - contine declaratii de variabile si declaratii de metode (numai antetul acestora).
- *Variabilele interfetei sunt implicit statice si finale si deci trebuie specificate valori initiale pentru acestea.*
- Metodele sunt declarate in interfata numai cu antetul lor, fara corp. Ele sunt implicit abstracte.

Interfete Java

Observatii:

1. Interfetele nu se pot instantia, adica nu se poate crea o instanta a unei interfete; deci, o interfata nu are o metoda constructor.

2. Nu pot exista implementari diferite pentru aceeasi metoda declarata in doua interfete.

- *Interfetele trebuie sa posede o protectie de pachet sau publica*.
- Interfetele care nu contin modificatorul *public* nu-si vor converti automat metodele la accesul *public* si *abstract* si nici constantele la accesul *public*.
- *O interfata nepublica are metode si constante nepublice*, acestea neputand fi folosite decat in clasele sau interfetele din acelasi pachet.
- Interfetele, ca si clasele, pot apartine unui pachet daca se foloseste instructiunea **package** in prima linie din fisierul-sursa.
- De asemenea, interfetele pot importa interfete sau clase din alte pachete.

Un exemplu de interfata este *Compara*:

```
/* INTERFATA folosita de mai multe clase pentru
   comparatii
* ------------------Metode publice--------------
* int comparaCu    --> Compara doua obiecte de tip
   Compara
* metoda trebuie definita in clasa care o implementeaza
*/
package clasegenerice;
public interface Compara
{
   int comparaCu(Compara rhs);
}
```

Interfete Java

- Interfata *Compara* declara o metoda pe care orice clasa derivata din ea (in declaratia clasei se foloseste cuvantul-cheie *implements*) trebuie sa o implementeze.

- Metoda *comparaCu ()* se va comporta similar cu metoda *compareTo()* din clasa *String*.

- Metoda este implicit publica si abstracta si deci nu trebuie sa folosim modificatorii de acces *public* si de metoda *abstract*.

Implementarea unei interfete

O clasa implementeaza o interfata in doi pasi:

1. declara interfata, folosind cuvantul-cheie *implements* in antetul sau

2. defineste implementari pentru toate metodele din interfata.

- Pentru a exemplifica implementarea unei interfete ne vom referi la clasa *FormaGeo* descrisa anterior.

- Aceasta clasa implementeaza interfata *Compara* si, deci, trebuie sa definim si metoda *comparaCu()* din interfata *Compara*.
- Implementarea metodei in clasa *FormaGeo* trebuie sa fie identica cu declaratia din interfata si, din acest motiv, metoda *comparaCu()* are ca parametru un obiect de tip *Compara* si nu un obiect de tip *FormaGeo*.
- Codul clasei este ilustrat in continuare (FormaGeo.java):

Implementarea unei interfete

```
/* Superclasa abstracta pentru forme */
 package formegeometrice;
 import clasegenerice.*;
 public abstract class FormaGeo implements
    Compara
 {
    private String nume;
    abstract public double arie();
    public FormaGeo(String numeForma) {
          nume = numeForma;
    }
```

Implementarea unei interfete

```
public int comparaCu(Compara rhs)  {
        if (arie() < ((FormaGeo) rhs).arie())  return -1;
        else
          if (arie() == ((FormaGeo) rhs).arie()) return 0;
                else    return 1;
    }
    final public String toString()  {
          return nume + ", avand aria " + arie();
    }
}
```

Implementarea unei interfete

Nota:

1. O clasa care implementeaza o interfata poate sa extinda si o alta clasa. In acest caz in declaratia clasei se foloseste mai intai cuvantul-cheie *extends* urmat de numele clasei din care deriveaza si apoi cuvantul-cheie *implements* urmat de numele interfetei pe care o implementeaza.
2. Retinem ca definirea metodelor declarate intr-o interfata trebuie obligatoriu sa fie facuta in fiecare clasa in parte, care implementeaza interfata respectiva.

Implementarea unei interfete

Nota:

3. Metodele interfetei definite in clasa care o implementeaza sunt mostenite de toate subclasele clasei respective. Aceste metode pot fi redefinite (suprascrise) in subclase.

4. Daca o clasa extinde o superclasa care implementeaza o interfata, atunci si clasa respectiva mosteneste interfata.

Implementarea unor interfete multiple

- O clasa poate sa implementeze mai mult decat o singura interfata.
- O clasa poate implementa mai multe interfete in doi pasi:

    1. declara toate interfetele pe care le implementeaza, folosind cuvantul-cheie *implements* in antetul sau urmat de numele tuturor interfetelor separate prin virgula;

    2. defineste implementari pentru toate metodele din toate interfetele pe care le implementeaza.
- Folosirea unor interfete multiple poate crea totusi complicatii.

Derivarea interfetelor

- Ca si in cazul claselor, *interfetele pot fi organizate intr-o ierarhie*.
- Atunci cand o interfata mosteneste o alta interfata, subinterfata primeste toate metodele si constantele definite in superinterfata.
- Pentru a deriva (extinde) o interfata se foloseste tot cuvantul-cheie *extends*, la fel ca in cazul definitiilor de clasa:

```
public interface <nume_interfata> extends
   <nume_superinterfata> {
.....
}
```

Derivarea interfetelor

Nota:

- In cazul interfetelor nu exista o radacina comuna a arborelui de derivare asa cum exista pentru arborele de clasa, clasa *Object*.
- *Interfetele pot exista independent sau pot mosteni o alta interfata*.
- Ierarhia de interfete este cu mostenire multipla.
- O singura interfata poate mosteni oricate clase are nevoie.

Întrebări?