Lucrarea 5

Interfeţe

Cuprins

Un obiect include date şi metode (operaţii) care permit modificarea datelor. Obiectul execută o operaţie atunci când primeşte o cerere (un mesaj) de la un client. Mesajele reprezintă singura cale prin care un obiect este determinat să execute o operaţie, iar operaţiile sunt singurul mod de a modifica datele interne ale obiectului. Din cauza acestor restricţii, starea internă a obiectului se spune că este încapsulată: ea nu poate fi accesată direct, iar reprezentarea ei este invizibilă dinspre exteriorul obiectului. Pentru fiecare operaţie declarată într-un obiect se precizează numele, parametrii şi valoarea returnată. Aceste elemente formează semnătura operaţiei.

Mulţimea tuturor semnăturilor corespunzătoare operaţiilor dintr-un obiect, accesibile clienţilor, reprezintă interfaţa obiectului. Aceasta descrie complet setul mesajelor care pot fi trimise spre obiectul respectiv. Un tip este un nume utilizat pentru a referi o anumită interfaţă. Un obiect este de tipul T dacă el acceptă toate mesajele corespunzătoare operaţiilor definite în interfaţa numită T.

Interfeţele sunt elemente fundamentale în sistemele OO. Obiectele sunt cunoscute doar prin intermediul interfeţelor lor. O interfaţa nu dă nici un detaliu relativ la implementarea unui obiect, iar obiecte distincte pot implementa diferit o aceeaşi cerere.

Interfeţe Java

Interfeţele Java reprezintă o colecţie de metode abstracte (doar semnătura, nu şi implementarea acestora) şi de constante. Utilizarea unei interfeţe este justificată în cazurile în care se doreşte ca o clasă să poată fi convertită în mai multe tipuri de bază între care nu există relaţie de moştenire. Deoarece *Java* nu suportă moştenirea multiplă, interfeţele reprezintă o variantă de rezolvare a acestei situaţii.

La fel ca şi o clasă, înainte de a putea utiliza o interfaţă ea trebuie declarată. Deoarece conţine metode abstracte, interfaţa nu poate fi instanţiată. Cu toate acestea, o variabilă poate fi declarată ca fiind de tipul interfeţei.

Declararea unei interfeţe în Java

O interfaţă Java este declarată prin cuvântul cheie *interface*. Structura generală a declaraţiei unei interfeţe este:

```
[modificator] interface id_interfata [extends lista_de_interfete]{
        listă_de_constante;
        lista_de_metode;
}
```

Modificatorul unei interfeţe poate fi *public* sau *abstract*. Având în vedere faptul că o interfaţă este abstractă prin definiţie, utilizarea modificatorului abstract este redundantă. *Spre deosebire de o clasă, o interfaţă poate moşteni mai multe interfeţe*. Moştenirea se face prin clauza *extends* urmată de lista interfeţelor moştenite (lista_de_interfete) separate prin virgulă. Prin derivare, o interfaţă moşteneşte de la interfaţa de bază atât metodele, cât şi constantele acesteia.

```
public interface Id1 extends Id2, Id3, Id4
```

În corpul interfeţei pot exista o listă de declaraţii de metode şi o listă de declaraţii de constante. *Metodele declarate într-o interfaţă sunt implicit abstracte*. Constantele(variabilele membre) declarate în interfaţă primesc atributele static şi final. Unui câmp final, după ce a fost iniţializat, nu i se mai poate modifica valoarea.

```
public interface Alergator{
        int lg_pista=1000;
        int timp_max=3;
        public void alearga();
        public void accelerează();
}
```

Metodele dintr-o interfaţă nu pot fi statice. Referirea constantelor se va face astfel:

```
        Alergator.lg_pista;
```

În urma compilării unui fişier sursă Java care conţine o interfaţă, va rezulta un fişier cu extensia *.class* ataşat interfeţei respective.

Utilizarea interfeţelor Java

Interfeţele Java pot fi utilizate în declaraţia unei clase, obligând clasa respectivă să implementeze metodele declarate în interfaţă. În Java, acest lucru e realizat prin folosirea cuvântului cheie *implements*:

```
class Sportiv extends Om implements Alergator, Saritor{
        …
        //implementarea metodelor din interfaţa Alergator;
        //implementarea metodelor din interfaţa Saritor;
        …\
}
```

Aşa cum se poate observa, o clasă poate implementa mai multe interfeţe. Dacă o clasă care implementează o interfaţă nu implementează toate metodele declarate în interfaţa respectivă, va trebui să fie declarată *abstractă*, altfel se va semnala eroare la compilare. Dacă o clasă *A* implementează o interfaţă *X*, un obiect al clasei *A* este de tip *X*, el putând să fie atribuit unui alt obiect de tip *X*. Numele unei interfeţe poate fi utilizat la fel ca şi numele unui clase, mai puţin la instanţierea prin operatorul *new*.

Exemplu de lucru cu interfeţe Java

Pentru a mai bună înţelegere a conceptelor prezentate în lucrare, vom recurge, în continuare, la un exemplu.

```
interface Cititor {
        public void citeste();
}
interface Scriitor{
        public void scrie();
}

interface Profesor extends Cititor, Scriitor{
        int nota_max=10;
        int nota_medie=5;
        int nota_min=1;
        public void apreciere();
}

interface Scufundator{
        int h=10;
        public void scufundare();
}

class Inotator implements Scufundator{
        static final int l=50;
        public void inot(){
            System.out.println("Inoata " + l + "metri ");
        }
        public void scufundare(){
            System.out.println("Scufunda " + h + "metri ");
        }
}

class ProfesorInot extends Inotator implements Profesor{
        public void citeste(){
            System.out.println("Citeste revista \"Aventuri la pescuit\" ");
        }

        public void scrie(){
            System.out.println("Nu scrie nimic!");
        }
        public void apreciere()
        {
            System.out.println("Slab. Primesti nota " +nota_medie);
        }
```

```
}

class ProfesorJava implements Profesor{
        public void citeste(){
            System.out.println("Citeste Thinking in Java");
        }
        public void scrie(){
            System.out.println("Scrie programe Java");
        }
        public void apreciere(){
            System.out.println("Excelent. Primesti nota " +nota_max);
        }
}
public class TestInterfata{
        public static void main(String [] args){
            ProfesorInot pi = new ProfesorInot();
            ProfesorJava pj = new ProfesorJava();
            ceCiteste(pi); ceScrie(pi);
            ceCiteste(pj); ceScrie(pj);
            cumApreciaza(pi); cumApreciaza(pj);
            pi.inot(); pi.scufundare(); //corect
            pj.inot(); pj.scufundare(); //incorect
        }
        public static void ceCiteste( Cititor c){
            c.citeste();
        }
        public static void ceScrie( Scriitor s){
            s.scrie();
        }
        public static void cumApreciaza( Profesor p){
            p.apreciere();
        }
}
```

Folosind referinţa de interfaţă se pot apela metodele enumerate în interfaţă, iar ceea ce se execută va fi codul corespunzător clasei "implementatoare", la care aparţine obiectul concret indicat de referinţă.

Asocierea operaţiilor cu obiectele

Când o cerere este trimisă unui obiect, operaţia care se va executa depinde de:

- cerere;
- obiectul care recepţionează cererea.

Obiecte diferite, care pot recepţiona cereri identice, pot avea implementări diferite ale operaţiilor care vor satisface cererile respective. Practic, prin cerere se identifică serviciul

dorit, iar prin obiectul receptor se alege o anume implementare a acelui serviciu. Asocierea dintre o operaţie solicitată şi obiectul care va pune la dispoziţie implementarea concretă a operaţiei se numeşte legare (*binding*). În funcţie de momentul în care ea se realizează, ***legarea*** poate fi:

- ***statică***: asocierea se realizează la compilare;
- ***dinamică***: asocierea se realizează la execuţie.

În programele Java, majoritatea apelurilor de metode presupun legare dinamică. Asocierea dinamică permite scrierea de programe în care:

- la emiterea unei cereri să nu ne preocupe ce obiect o va recepţiona, ştiindu-se că orice obiect a cărui interfaţă include o semnătură potrivită va fi bun;
- obiecte având interfeţe identice pot fi substituite unul altuia, la execuţie; această substituire se mai numeşte polimorfism.

Temă

1. Se cere să se modeleze activitatea mai multor ghişee bancare. Sistemul este format din următoarele entităţi, cu atributele corespunzătoare:

Banca

- denumire_banca (*String*)
- clienţi (colectie de obiecte de tip *List<Client>*)

Client

- nume (*String*)
- adresa (*String*)
- conturi (colectie de obiecte de tip *Set<ContBancar>*; un client trebuie să aibă cel puţin un cont, dar nu mai mult de cinci conturi)

ContBancar

- numarCont (String)
- suma(float)
- moneda (String)
- data_deschiderii (Calendar)
- data_ultimei_operatiuni (Calendar)

Conturile bancare pot fi în *RON* şi în *EUR*. Pentru conturile in *RON*, dobânda se calculează astfel: *0,3 RON* pe zi pentru sume sub *500 RON* si *0,8 RON* pe zi pentru sume mai mari.

Conturile in *EUR* au o dobândă de *0,1 EUR* pe zi.

Toate conturile implementează o interfaţa ***Operatiuni*** care are metodele:

public float calculeaza_dobanda() – calculează numărul de zile scurs de la data ultimei operaţiuni asupra contului şi până la data curentă. Numărul de zile obţinut este utilizat pentru calculul dobânzii după cum este menţionat mai sus

public float actualizare_suma() – adaugă la suma din cont dobânda corespunzătoare numărului de zile scurs, actualizează data ultimei operaţiuni si returnează suma

public void depunere(float suma) – actualizează suma, data ultimei operaţiuni şi adună la suma existentă suma depusă

public void extragere (float suma) – actualizează suma, data ultimei operaţiuni şi scade din suma existentă suma depusă

Toate clasele vor avea constructori şi metode de prelucrare şi de afişare a datelor membre. Se va redefini metoda *toString()* în fiecare clasă în aşa fel încât să returneze un *String* care reprezintă valorile variabilelor membre din acea clasă.

De asemenea, se va crea o clasă *ClientiiBancilor*, care conţine programul principal ce oferă următorul meniu:

1. Adăugare bănci, clienţi, conturi. Băncile vor fi introduse într-o colecţie de obiecte de tip *Vector<Banca>* (se va utiliza clasa Vector).
2. Afişare date introduse
3. Depunerea unei sume într-un cont
4. Extragerea unei sume dintr-un cont
5. Transfer de bani între două conturi

Datele pot fi preluate de la tastatură sau din fişier.

2. În fișierul de intrare *in.txt* se găsesc informații despre cărți, precum în captura de ecran de mai jos.

```
1; Plansul lui Nietzsche; Irvin Yalom; 1992
2; De veghe in lanul de secara; J.D. SALINGER; 1951
3; Mitul lui Sisif; Albert Camous; 1942
```

Pentru fiecare carte se reține id-ul cărții (număr unic de identificare), titlul cărții, numele autorului şi anul apariției.

Să se creeze o clasă *Carte* cu variabile membre *titlul, autorul* şi *anul apariție*. Clasa va dispune de constructor cu parametri, *gettere* si *settere* pentru accesarea variabilelor membre şi va redefini metoda *toString()* astfel încât aceasta să returneze un *String* cu valorile variabilelor membre separate prin virgule.

Să se creeze o colecție de obiecte de tip *Map<Integer,Carte>* în care se vor adăuga cărțile citite din fișier. *Id-ul* cărților reprezintă cheia elementelor din colecția *Map*, iar valoarea elementelor din colecție este reprezentată de obiecte de tip *Carte.* Se va utiliza implementarea *HashMap* a interfeței *Map*. Pentru colecția *Map* creată să se implementeze următoarele cerințe:

1. Să se afișeze colecția (se vor afișa atât cheile cât şi valorile).
2. Să se creeze o colecție *List<Carte>* cu valorile din colecția *Map*. Să se afișeze colecția creată.
3. Să se ordoneze elementele din colecția *List* după titlul cărți şi să se afișeze colecția.