Pachete*

Un **pachet** este o colecție de clase și interfețe care poate avea asociat un nume.

Pachetele se utilizează pentru a grupa clasele pe funcționalități, pentru a putea accesa clase din directoare diferite, pentru a evita conflicte de nume (dacă există în aplicație clase cu același nume), pentru a controla accesul la anumite clase și la membrii acestora.

În primele laboratoare unitățile de compilare au fost plasate toate în același director, fără a fi specificat explicit faptul că clasele aparțin unui pachet (fără a folosi instrucțiunea `package`). Atunci ia naștere un **pachet fără nume**, constituit din aceste unităţi.

Pentru aplicații mai complexe, în care plasarea claselor în acelașii director nu este o soluție mulțumitoare, apare necesitatea grupării claselor după anumite criterii, în directoare diferite. Sunt necesare astfel pachete cu nume, pentru a putea accesa clase din alte directoare.

Vom prezenta pe scurt în cele ce urmează cum putem grupa clase în pachete cu nume.

Pachetele sunt organizate în **directoare** în sistemul de fişiere al maşinii gazdă, **numele pachetului reflectând această structură de directoare**.

Un pachet poate conţine:
- subpachete ale pachetului;
- tipuri (clase sau interfețe) declarate în unităţile de compilare ale pachetului (având toate aceeaşi declarare **`package`**).

Pentru a crea în directorul de lucru un pachet, al cărui nume dorim să fie `P1`, trebuie creat un subdirector cu numele `P1` şi apoi introduse în subdirector unităţile de compilare ale pachetului.

Fiecare unitate de compilare din pachetul `P1` trebuie să înceapă cu declararea:

```
package P1;
```

care dă informaţii compilatorului asupra numelui pachetului din care face parte unitatea de compilare.

Exemplu Clasa "principală" `Main` crează un obiect `ob` de tipul unei clase `C` şi invocă prin el o metodă `suma` a acestei clase.

Vom plasa cele două clase **în subdirectorul `Pachet` al directorului curent,** prevăzând în ambele fişiere directiva:

```
package Pachet;
```

Main.java

```
package Pachet;
class Main {
  public static void main(String[] sss) {
    int a=1, b=2;
    C ob = new C();
    System.out.println( ob.suma(a,b) );
  }
}
```

C.java

```
package Pachet;
class C {
  int suma(int x, int y) {
      return x+y;
  }
}
```

Compilarea se face:

- din directorul curent prin: `javac Pachet\Main.java`
- sau din directorul `Pachet` prin: `javac -classpath .. Main.java`

iar executarea se face:

- din directorul curent prin: `java Pachet.Main`
- sau din directorul `Pachet` prin: `java -classpath ..;. Pachet.Main`

Trecem la prezentarea generală.

Dacă o unitate de compilare face referire la o clasă `C` din pachetul `P1`, atunci în această unitate trebuie plasată declararea `import`:

```
import P1.C;
```

Dacă dorim acces la toate clasele pachetului, putem folosi următoarea formă pentru declarare:

```
import P1.*; //! Prin import nu se importă şi subpachete
```

Structura de pachete este ierarhică (formează un arbore); un pachet `P1` poate avea un subpachet `P2`. Pentru crearea subpachetului `P2` trebuie creat în directorul `P1` subdirectorul `P2`. O unitate de compilare din subpachetul `P2` trebuie **să înceapă** cu declararea:

```
package P1.P2;
```

Observăm că numele subpachetului reflectă structura de directoare, începând de la rădăcină. Această structură trebuie urmată şi în declarările import

```
import P1.P2.*;
```

O clasă sau interfaţă dintr-un pachet poate fi accesată din afara pachetului doar dacă are modificatorul `public`. Aceeaşi regulă este valabilă pentru accesarea directă a membrilor unei clase a pachetului. Pentru constructori trebuie procedat la fel, afară de cazul în care se foloseşte constructorul implicit, care este considerat ca având modificatorul `public`.

Exemplu Să considerăm o aplicație care reunește într-un meniu câteva dintre exemplele date în laboratoarele trecute, grupate pe temele de care se leagă exemplele. Aplicația are următoarea structură de directoare

```
exemple
      clase
            citire
      meniu
```

Directorul `exemple` conține pachetul `clase` (care va conține exemple legate de clase în Java în general), care are ca subpachet `citire` (care va conține clase care ilustrează citirea de la tastatură și din fișier în Java). De asemenea, directorul `exemple` conține directorul `meniu` (cu clasa principală, care permite utilizatorului să selecteze ce exemple legate de limbajul Java dorește să ruleze).
Pentru simplitate, fiecare pachet va conține cel mult două clase.
Astfel, în `clase` se află unitatea de compilare `Dreptungi.java`:

Dreptungi.java

```
package clase;
public class Dreptunghi{
    double lung,lat;
    public Dreptunghi(){
      lung=1;
      lat=1;
    }
    public Dreptunghi(double lung1,double lat1){
        lung=lung1;
        lat=lat1;
    }
    public double arie(){
        return lung*lat;
    }
    public boolean maiMare(Dreptunghi d){
        return arie()<d.arie();
    }
    public String toString(){
      return "lungime "+lung+",latime "+lat;
    }
}
```

În subdirectorul `citire` al directorului `clase` se află unitățile de compilare `Muchie.java` și `TestScanner.java`

Muchie.java

```
package clase.citire;
class Muchie{
	int vi,vf;
	double cost;
	Muchie(int vi,int vf, double cost){
		this.vi=vi;
		this.vf=vf;
		this.cost=cost;
	}
	public String toString(){
		return "("+vi+","+vf+")"+" "+cost;
	}
}
```

TestScanner.java

```
package clase.citire;
import java.util.*;
import java.io.*;
public class TestScanner{
	static final int NR_MAX=3;
		public void fisier(){
		int i,n;
		Muchie muchii[];
		try{
			Scanner scFisier=new Scanner(new File("muchie.txt"));
			n=scFisier.nextInt();
			muchii=new Muchie[n];
			for(i=0;i<n;i++)
			muchii[i]=new Muchie(scFisier.nextInt(),
scFisier.nextInt(), scFisier.nextDouble());
			scFisier.close();
			for(i=0;i<n;i++)
				System.out.println(muchii[i]);
		}
		catch(FileNotFoundException fnf){
			System.out.println("Fisier inexistent.");
		}

	}

}
```

În meniu se află unitatea de compilare `Meniu.java` și fișierul `muchie.txt`:

Meniu.java

```
package meniu;//poate lipsi
import instructiuni.ExpSwitch;
import clase.*;//nu importa si subpachetul citire
import clase.citire.TestScanner; //nu trebuie si Muchie
import java.util.*;

public class Meniu{
	public static void main(String arg[]){
		int optiune=0;
		Scanner sc=new Scanner(System.in);
		do{
			System.out.println("Alegeti o optiune");
			System.out.println("1-Clase");
			System.out.println("2-Clasa Scanner");
			System.out.println("0-Iesire");
			optiune=sc.nextInt();
			switch(optiune){
				case 1:
					Dreptunghi d=new Dreptunghi(3,4);
					System.out.println(d+" arie "+d.arie());
					//System.out.println(d.lung); //- NU
					break;
				case 2:
					//Muchie m; //-NU
					TestScanner ts=new TestScanner();
					sts.fisier();
			}

		}while(optiune!=0)	;


	}
}
```

muchie.txt

```
5
2 3 10
1 3 4
1 2 17
1 4 20
3 4 5
```

Pentru a compila și rula această aplicație, trebuie să înțelegem unde sunt căutate clasele la compilarea și rularea unei aplicații.

La executarea unor comenzi ale maşinii virtuale Java, ca de exemplu javac şi java, are loc o **căutare de clase**, adică de fişiere cu extensiile .java şi .class. Pentru precizarea locului unde vor fi căutate aceste clase, Java prevede ***variabila de mediu* CLASSPATH.** Valoarea variabilei este o mulţime de căi separate între ele prin caracterul ';'.

Variabilei CLASSPATH i se poate atribui o valoare prin comanda:

SET CLASSPATH=*path*

sau prin opţiunea `classpath` din comenzile `java` şi `javac`.

Ordinea în care apar căile determină ordinea de căutare a claselor.

Executarea comenzii `javac` are ca efect compilarea uneia sau a mai multor unităţi de compilare, ce conţin definiţii de clase sau interfeţe. Sunt compilate toate tipurile (clase sau interfeţe) din unităţile respective, *dar şi toate tipurile referite din ele şi care sunt regăsite în calea pentru cod*.

O formă restrânsă a comenzii este:

`javac` *`opt unit`*

unde atât opţiunile din lista *`opt`*, cât şi unităţile de compilare din lista *`unit`* sunt separate între ele prin blancuri.

Dintre opţiuni vom menţiona numai următoarele două:

`-d` *`dir`*

`-classpath` *`path`* sau, prescurtat: `-cp` *`path`*

Prima opţiune permite plasarea fişierelor byte-code, rezultate în urma compilării, în directorul *`dir`*; în lipsa acestei opţiuni, drept *`dir`* este considerat directorul curent.

A doua opţiune permite precizarea unei noi căi de cod, în care se va încerca regăsirea (căutarea) claselor. Căutarea include atât fişierele cu extensia `.java`, cât şi cele cu extensia `.class`.

La căutarea unui tip (clasă sau interfaţă) `tip` deosebim trei cazuri:

1) este găsit doar `tip.class`: este folosit acest fişier byte-code;

2) este găsit doar `tip.java`: este compilat acest tip şi este folosit rezultatul compilării;

3) sunt găsite atât `tip.class` cât şi `tip.java`: este folosit `tip.class` doar dacă acest fişier este mai recent decât `tip.java`.

Atribuirea unei valori variabilei CLASSPATH anulează atribuirea precedentă. Astfel, pentru a include directorul curent în cale trebuie inclus explicit caracterul '.' (caracterul '.' semnifică directorul curent, iar '..' directorul părinte al directorului curent).

Precizările de mai sus sunt valabile şi pentru setările prin opţiunea `classpath` din comenzile `javac` şi `java`, cu observaţia că în acest caz noua setare este valabilă numai pentru comanda respectivă.

În cele ce urmează, vom utiliza opțiunea `classpath` a comenzilor (nu variabila de mediu CLASSPATH, cele două moduri de lucru fiind similare).

Compilarea aplicației din directorul exemplu

Să presupunem că directorul `exemplu` este pe partiția `c` (`c:\exemple`).

Presupunem că **ne aflăm în directorul `c:\exemple\meniu`** (în care se găsește clasa principală, cu metoda `main`) (Pentru a ajunge în acest director din linia de comanda se executa `cd c:\exemple\meniu`)

La compilare, variabila `classpath` trebuie să conțină **calea către rădăcina structurii de pachete** care în acest caz este părintele directorului curent (adică directorul `c:\exemple`) și **directorul curent** (dacă sunt utilizate clase din acest director); este suficient să compilăm clasa prinicipală (care conține metoda `main`), fiind compilate automat unitățile de compilare utilizate în clasa principală:

```
javac –classpath .;.. Meniu.java
```

Rularea aplicației din directorul exemplu

La rulare variabila `classpath` trebuie să conțină calea către rădăcina structurii de pachete și directorul curent

```
java –classpath .;.. Meniu
```

ⓘ **Observație** Dacă includem și clasa `Meniu` în pachet (decomentăm instrucțiunea `package` meniu de la începutul fișierului `Meniu.java`), atunci interpretorului i se specifică numele complet al clasei (prefixat cu numele pachetului). Rularea se va face în acest caz cu comada

```
java – classpath .;.. meniu.Meniu
```

🕮 Exerciţii:

1. Adăugaţi clase noi în pachetele din exemplul anterior (de exemplu clasa `Complex` în pachetul clase). Adăugaţi şi un subpachet `siruri` pentru pachetul clase care să conţină o clasă care exemplifică lucrul cu clasa `String`. Adăugaţi opţiuni în meniu corespunzătoare claselor adăugate.

2. Compilaţi şi rulaţi aplicaţia din directorul `c:\exemple` (în loc de `c:\exemple\meniu`).

3. Compilaţi şi rulaţi aplicaţia din orice alt director.

4. Considerăm clasele `Persoana` şi `Angajat`, derivată din `Persoana`.

Urmăriţi aspectele prezentate la curs (`this`, `super`, metode şi câmpuri moştenite, modificatori de acces, legare dinamică).

🗋 Persoana.java

```
class Persoana{
	public String nume;
	private int varsta;

	public Persoana(){}
	public Persoana(String nume){
		this.nume=nume;
	}
	public Persoana(String nume, int varsta){
		this(nume);//!! Apel constructor
		this.varsta=varsta;
	}
	public int getVarsta(){
		return varsta;
	}
	public void setVarsta(int varsta){
		if(varsta>0)
			this.varsta=varsta;
	}
	public int compara(Persoana p){
		int dif=varsta-p.getVarsta();
		if(dif>0) return 1;
		if(dif<0) return -1;
		return 0;
	}
	public String toString(){
		return nume+": "+varsta+" ani";
	}
}
```

Angajat.java

```
class Angajat extends Persoana{
	private double salariu;
	int vechime;
	//mosteneste nume

	Angajat(){//implicit super();
	}
	Angajat(String numeIn){
		nume=numeIn;//implicit super();
	    //SAU direct super(numeIn);
	}
	Angajat(String nume, int varsta){
		super(nume,varsta);
	}
	Angajat(String numeIn, int varstaIn, int salariuIn, int vechimeIn){
		super(numeIn,varstaIn);
		setSalariu(salariuIn);
		vechime=vechimeIn;
		//varsta=varstaIn; // nu se poate, varsta nu este mostenit
fiind private
	}

	double getSalariu(){
		return salariu;
	}
	void setSalariu(int salariu){
		this.salariu=salariu;
	}
	public int compara(Angajat p){//supraincarcare, nu rescriere
		int dif=vechime-p.vechime;
		if(dif>0)
			return 1;
		if(dif<0)
			return -1;
		return 0;
	}
	public String toString(){
		return super.toString()+" - vechime "+vechime;
	}

	public double getSpor(){
		return 0.04*vechime/3;
	}
}
```

Main.java

```
class Main{
	public static void main(String arg[]){
		Angajat a1=new Angajat("Ionescu");
		a1.vechime=5;
		a1.setVarsta(50);//metoda mostenita de la Persoana
		a1.setSalariu(1000);
		System.out.printf("%s are spor de vechime %.2f",a1,a1.getSpor());
		System.out.println();

		Persoana p1,p2;
		p1=a1;//a1 privit ca persoana
		a1.setVarsta(55);
		System.out.println(a1); //a1 afisat ca Angajat, deoarece se
foloseste metoda toString() a tipului real

		Angajat a2=new Angajat("Popescu", 35, 1500,10);
		System.out.println(a2);
		if(a1.compara(a2)>0)//metoda compara din Angajat
			System.out.println(a1.nume+" are vechime mai mare decat
"+a2.nume); //nume-mostenit
		else
			if(a1.compara(a2)==0)
			    System.out.println(a1.nume+" are aceeasi vechime cu
"+a2.nume);
			else
			    System.out.println(a1.nume+" are vechime mai mica
decat "+a2.nume);

		p1=a1;
		p2=a2;
		if(p1.compara(p2)>0)
//metoda compara din Persoana, mostenita si in Angajat
			System.out.println(p1.nume+" are varsta mai mare decat
"+p2.nume);
		else
			if(p1.compara(p2)==0)
				System.out.println(p1.nume+" are aceeasi varsta
cu "+p2.nume);
			else
				System.out.println(p1.nume+" are varsta mai mica
decat "+p2.nume);
	}
}
```

Comentariu

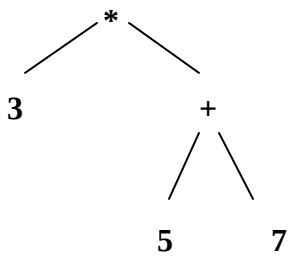
Metoda `compara(Angajat)` din clasa `Angajat` nu rescrie metoda `compara(Persoana)` moştenită din clasa `Persoana` deoarece nu au aceeeaşi signatură, este o metodă nouă cu acelaşi nume cu cea moştenită (supraîncarcare).

5. Adăugaţi clasele `Persoana` şi `Angajat` într-un pachet angajati, iar clasa `Main` lăsaţi-o în directorul de lucru, adăugându-le instrucţiuni de import corespunzătoare. Testaţi exemplul cu această structură şi explicaţi de ce anumite secţiuni de cod nu mai sunt corecte datorită modificatorilor de acces. Încercaţi să schimbaţi modificatori de acces şi structurile de pachete pentru a înţelege mai bine semnificaţia modificatorilor de acces.

Probleme

1. Daţi un exemplu legate de extinderi de clase în care să fie evidenţiate ideea de moştenire, rescrierea metodelor statice şi nestatice, polimorfismul şi legarea dinamică, modificatori de acces, utilizarea cuvântului cheie `super`.
2. Se dă o expresie care conţine constante şi operatorii binari +, -, *, / **în forma prefixată**. Să se construiască arborele binar asociat expresiei, să se evalueze expresia folosind acest arbore şi să se afişeze expresia în forma iniţială cu paranteze (în formă infixată) şi în forma postfixată. De exemplu, pentru expresia 3 * (5 + 7) forma prefixată este * 3 + 5 7, iar forma postfixată este 3 5 7 + *.

   Arborele binar asociat expresiei este



   Modificaţi programul astfel încât expresia să poată conţină şi variabile. La evaluarea expresiei se vor cere valori pentru fiecare variabilă (o singură dată).