Clase, metode (constructori, mutatori, accesori, destructor (finalize), metode membre), specificatori de acces, Interfete, Expresii lambda

Classes, methods (constructors, setters, getters, destructor (finalize), member methods), access specifiers, Interfaces, Lambda expressions

Obiective:
- implementarea claselor și interfețelor Java, specificatori de acces
- clase interioare, clase anonime
- metode membre în clase
- expresii lambda

Objectives:
- implementing Java classes and interfaces, access specifiers
- inner classes, anonymous classes
- member methods
- lambda expressions

Clase Java

O clasă Java se declară folosind cuvântul cheie *class* urmat de numele clasei.
O clasă poate conţine variabile şi metode (implementate sau declarate). Aceştia poartă numele de membri ai clasei.
Atât clasa cât şi membrii săi pot fi marcate folosind specificatorii de acces/vizibilitate.

Sintaxa:

```
[specificatori_de_acces] class NumeClasa{
        //implementarea clasei
}
```

Specificatorii de acces pot lipsi (se considera default) sau pot fi *public* şi *abstract* sau *final*. Clasele interioare pot avea și alți specificatori de acces.

Notă: dacă o clasă este declarată ca fiind publică, fişierul în care este stocată trebuie să poarte acelaşi nume cu numele clasei. Dacă clasă nu este publică, nu este nici un fel de restricţie cu privire la numele fişierului sursă.

În limbajul Java exista o serie de restricţii şi de recomandări cu privire la modul de numire a claselor.

Restricţii

- Numele de clase nu pot fi cuvinte cheie predefinite sau nume de entităţi Java (existente sau definite de programator).
- Numele de clase sunt *case-sensitive*.
- Numele de clase sunt formate dintr-un singur cuvânt. Un cuvânt este orice secvenţă oricât de lungă de caractere Unicode, fără spaţii.
- Numele de clase pot începe cu orice literă sau cu caracterul *$*.

Recomandări

- Numele claselor trebuie să înceapă cu majusculă.
- Dacă numele unei clase este format din mai multe cuvinte, recomandarea ar fi ca separarea între acestea să se facă capitalizând fiecare iniţială a cuvintelor componente. Sunt acceptate (dar nerecomandate) şi alte caractere ca separatori (de ex. caracterul _).

Specificatori de acces/vizibilitate

Limbajul de programare Java prevede următorii specificatori de acces/vizibilitate, care pot fi aplicaţi (după caz) claselor metodelor sau variabilelor.

final: poate fi aplicat claselor, metodelor şi variabilelor. Semnificaţia acestui cuvânt cheie este legată de imposibilitatea de a modifica entitatea Java care a fost marcată ca fiind final. Astfel, o clasă final nu poate fi derivată. O metodă final nu poate fi suprascrisă. O variabilă final devine constantă.

abstract: poate fi aplicat claselor şi metodelor. O metodă trebuie marcată cu acest cuvânt cheie dacă nu are implementare ci doar declarare. O clasă care are cel puţin o metodă abstractă trebuie marcată ca fiind abstractă. Mecanismul nu este implicit, aşa cum era cazul în limbajul C/C++.

static: poate fi aplicat variabilelor, metodelor şi blocurilor de cod. O variabilă statică este asociată clasei din care face parte şi nu unei anumite instanțe. O variabilă static poate fi accesată fie prin intermediul unui obiect, fie direct prin numele clasei de care aparţine. O metodă statică este de asemenea asociată clasei din care face parte şi nu unei anumite instanțe. O metodă statică nu poate accesa alţi membri nestatici ai clasei din care face parte. Metodele statice nu au acces la variabilă implicită this.

native: poate fi aplicat doar metodelor. Semnificaţia acestui cuvânt cheie semnifică faptul că implementarea metodei este localizată în altă parte, şi anume chiar în exteriorul JVM, într-o bibliotecă.

transient: se poate aplica doar variabilelor. O variabilă de acest tip nu este considerată ca făcând parte din starea persistentă a unui obiect. Dacă obiectul este serializat, variabilele tranzitive nu vor fi regăsite în reprezentarea binară a obiectului.

synchronized: este folosit pentru a controla accesul mai multor fire de execuţie la resursele comune. Mai multe detalii sunt prezentate în secţiunea 12 unde se analizează mai în detaliu mecanismul de programare concurenta în Java.

volatile: poate fi aplicat variabilelor şi semnifică faptul că acestea pot fi modificate în mod asincron şi de aceea compilatorul îşi ia măsuri de precauţie speciale. Acest specificator de acces este destul de puţin folosit.

public: poate fi aplicat claselor, variabilelor şi metodelor. O clasă publică este vizibilă de oriunde din exterior (unde este vizibil pachetul din care face parte, dacă clasa e asociată unui pachet). Un membru public (variabilă sau metoda) poate fi accesat de oriunde din exteriorul clasei, prin intermediul instanţelor create.

protected: poate fi aplicat variabilelor sau metodelor. Membrii marcaţi astfel pot fi accesaţi de oriunde din acelaşi pachet din care face parte clasa de care aparţin şi de oricare din subclasele (indiferent de pachetul din care fac parte) clasei în care sunt declaraţi. Membrii protejaţi nu pot fi accesaţi direct din exteriorul clasei prin intermediul instanţelor.

default: nu este un cuvânt cheie Java şi reprezintă atributul implicit care este aplicat claselor, metodelor sau variabilelor care nu au specificat nici unul din atributele public, private sau protected. Tot ce poartă acest specificator poate fi accesat de oriunde din pachetul din care face parte clasa sau membrii acesteia.
private: poate fi aplicat variabilelor sau metodelor unei clase. Este cel mai restrictiv specificator de vizibilitate, iar membrii privaţi pot fi accesaţi doar din interiorul clasei de care aparţin.

Metode (funcţii) Java

O metodă (funcţie) Java este o entitate software care grupează o secvenţă de declaraţii de variabile, instrucţiuni şi/sau blocuri de program şi poate returna sau nu o variabilă de un anumit tip.

Sintaxa:

```
[specificatori_de_acces] tip_returnat numeMetoda (lista_de_parametri){
        //implementare metoda
}
```

Tipul returnat poate fi void în cazul în care metoda nu returnează nimic sau orice alt tip de date (elementare sau nu), de tip sir sau nu. În Java este permisă returnarea din funcţii a variabilelor de tip sir datorită faptului că în acest limbaj de programare nu există pointeri ca în C/C++.
Numele metodei este o secvenţă alfa-numerică de caractere.
Lista de parametri este o înşiruire de tipuri şi nume de variabile, separate prin virgulă, pe care funcţia în cauză le primeşte în momentul apelului.

Convenţii de numire a metodelor în Java

Restricţii

· Numele de metode nu pot fi cuvinte cheie predefinite său nume de entităţi Java (existente sau definite de programator).
· Numele de metode sunt case-sensitive.
· Numele de metode sunt formate dintr-un singur cuvânt. Un cuvânt este orice secvenţă oricât de lungă de caractere Unicode, fără spaţii.
· Numele de metode pot începe cu orice literă sau cu caracterul $.

Recomandări

· Numele metodelor trebuie să înceapă cu minusculă.
· Dacă numele unei metode este format din mai multe cuvinte, recomandarea ar fi ca separarea între acestea să se facă capitalizând fiecare iniţială a cuvintelor component (mai puţin primul). Sunt acceptate (dar nerecomandate) şi alte caractere ca separatori (de ex. caracterul _).

Tot în această secţiune se pot defini termenii:
- metoda mutator: metoda (publică, de regulă) care primeşte ca parametri nişte valori pe care le asociază variabilelor corespondente din clasă; prototipul metodelor mutator urmează tiparul: *public void setNumeVariabila(tip_variabila noua_valoare)*;

- metoda accesor: metoda (publică, de regulă) care returnează (sau afişează, practică mai puțin specifică acestui context) valoarea unor variabile din clasă; prototipul metodelor accesor au următorul șablon de declarare: *public tip_variabila getNumeVariabila()*;

Metodele constructor și destructor

Există o categorie aparte de metode membre ale unei clase, şi anume ***metodele constructor***. Acestea se deosebesc de restul metodelor prin faptul că:

- au numele identic cu cel al clasei din care fac parte;
- nu specifică nici un tip returnat (nici măcar *void*);

Prin analogie cu limbajul C++, Java pune la dispoziţie un constructor implicit vid şi fără parametri. Acesta serveşte la crearea obiectelor în modul în care s-a văzut într-un exemplu anterior. Clasa *MyClass* a putut fi instanțiata (*MyClass ob1 = new MyClass();*) deşi nu a fost specificat nici un constructor explicit vid.

În momentul în care se declară şi implementează cel puţin un constructor explicit (cu sau fără parametri), constructorul implicit dispare.

În ceea ce privește ***metodele destructor*** specifice unei clase, limbajul Java (spre deosebire de C++) nu pune la dispoziție nici un mecanism care să suporte aceste entități software. Cu alte cuvinte, în Java nu există destructori.

Există însă un mecanism complementar care permite executarea unei secvențe de cod în momentul distrugerii unui obiect. Acest mecanism se bazează pe metoda

public void finalize()

care este prezentă implicit în fiecare clasă Java fiind definită în clasa *Object* și moștenită implicit. Suprascrierea acestei metode ține loc de destructor.

Clase interioare

Java permite definirea unor clase în cadrul altor clase. Astfel de clase sunt numite clase interioare. O clasă interioară este membră a clasei în care a fost declarată.

Sintaxa:

```
class ClasaExterioara{
    //...
    [static] [specificator_de_vizibilitate] class ClasaInterioara{
        //...
    }
}
```

Ca membră în clasa din care face parte, o clasă interioară are acces la toţi membrii clasei care o încapsulează, indiferent de specificatorul de acces (*public*, *protected*, *private*).

O clasă interioară poate fi marcată cu oricare din specificatorii de vizibilitate *public*, *protected* sau *private*.

O instanţă a unei clase interioare este asociată întotdeauna cu o instanţă a clasei încapsulatoare şi are acces direct la membrii acesteia. Pentru a instanția o clasă interioară se foloseşte o sintaxă similară cu cea prezentată mai jos.

Sintaxa:

```
ClasaExterioara obiect_exterior = new ClasaExterioara();
ClasaExeterioara.ClasaInterioara obiect_interior = obiect_exterior.new ClasaInterioara();
```

Clasele interioare pot fi marcate şi cu specificatorul de acces *static*. O clasă interioară static a are acces doar membrii statici din clasa încapsulatoare. O clasă interioară statică este asociată cu însăşi clasa încapsulatoare şi nu cu o anumită instanţă a acesteia şi de aceea modalitatea de a accesa membrii acesteia ia în considerare numele ambelor clase:

Sintaxa:

ClasaExterioara.ClasaInterioara.metoda();

Instanțierea claselor interioare statice foloseşte o sintaxă similară cu cea de mai jos:

Sintaxa:

ClasaExterioara.ClasaInterioara obiect_interior = new ClasaExterioara.ClasaInterioara();

Clase abstracte

O clasă Java trebuie marcată cu cuvântul cheie *abstract* dacă are cel puţin o metodă abstractă (fără implementare, în corpul clasei apare doar declararea metodei).

Sintaxa:

```
abstract class NumeClasa{
        public abstract void metodaAbstracta();
        public void metodaNormala(){
                //implementarea metodei
        }
}
```

Clasele abstracte nu pot fi instanțiate. Clasele abstracte pot fi doar moştenite. Dacă clasă care moşteneşte o clasă abstractă nu implementează toate metodele abstracte moştenite, devine la rândul ei clasa abstractă.

Interfeţe Java

Entităţile software numite interfeţe sunt specifice limbajului de programare Java (nu au echivalent în C/C++).
O interfaţă Java se defineşte folosind cuvântul cheie interface.

Sintaxa:

```
[specificator_de_acces] interface NumeInterfata{
        //corp interfaţa
}
```

Regulile referitoare la specificatorul de acces sunt aceleaşi ca şi în cazul claselor. O interfaţă poate fi marcată ca fiind public sau default (nu se specifică nimic în poziţia specificatorului de acces).

Interfeţele Java se stochează în fişiere sursa *.java. Dacă interfaţa este publică, numele fişierului trebuie să coincidă cu cel al interfeţei (analog cu clasele publice).

O interfaţă Java poate conţine:

- variabile marcate cu atributele *static* şi *final* (constante statice); omiterea acestor specificatori de vizibilitate este permisă, mediul de programare marcându-i implicit în acest mod;
- declaraţii (prototipuri) de metode, fără implementare
- metode marcate cu cuvântul cheie *default*

Toţi membrii interfeţelor sunt consideraţi impliciţi ca fiind publici. Deşi interfeţele conţin doar declaraţii de metode, nu este necesară folosirea specificatorului *abstract*.

Expresii lambda

Apărute ca alternativă la interfețele funcționale, pt. a elimina folosirea claselor anonime și a scrierii verticale a codului. Pot fi considerate ca fiind metode anonime.

Sintaxă:

(listă_de_parametri) -> corp

- listă de parametri - parametri ([tip] și nume, separați prin ,)

- corp - un singur bloc funcțional, delimitat sau nu prin {}

- se pot specifica valori returnate

Lucru individual

1. Scrieți o aplicație care definește o clasă denumită Copil . Definiți metodele și variabilele membre ale acestei clase pentru următoarele acțiuni/caracteristici.
- numele copilului
- data nașterii
- copilul știe să se prezinte: ”Salut, numele meu este ...”
- copilul știe să spună câti ani are
- copilul știe să adune două numere mai mici decât 10 și să returneze rezultatul sub forma: ”Suma lui X și Y este Z”
- copilul poate să spună alfabetul în forma directă, cât și inversă
- copilul știe să spună “La revedere”
- copilul știe să coloreze o tablă de șah de dimensiune dată folosind culori alternative (pentru culori folosiți simbolurile 1 și 0)
- copilul știe să joace X-0 singur ☺ (!!! Folosiți aplicația realizată în tema anterioară)

Precizări: numele copilului și data sa de naștere nu pot fi accesate din afara clasei.

Toate informațiile despre un Copil vor fi completate cu ajutorul unui obiect de tip Copil și variabilele și metodele corespunzătoare acestui obiect. Interacțiunea cu un copil se va face tot prin intermediul unui obiect instanțiat în main.

==========

2. Fie o interfață Grup ce definește următoarele metode: adăugare, eliminare, verificarea prezenței unui persoane în grup, listarea tuturor persoanelor prezente în grup la un moment dat și listarea în ordine alfabetică a persoanelor cu prenume distinct din grup. Interfața are o variabilă statică ce reține numărul de persoane aflate în grup. Când o persoană părăsește grupul, aceasta trebuie să-și ia la revedere de la grup.

Implementați interfața într-o clasă denumită GrupCopii și folosiți clasa Copil definită anterior.

==========

3. Creați un joc de tip Minesweeper. Toate funcționalitățile jocului vor fi implementate în clasa GameMinesweeper. Constructorul va avea ca și elemente de inițializare dimensiunea tablei de joc și numărul de mine ascunse. Numărul de mine nu poate să fie mai mare decât o treime din numărul total de căsuțe de pe tabla de joc.
Poziția minelor va fi generată aleator. Fiecare căsuță de pe tabla de joc va conține 0 pentru o căsuță neminată, 1 – pentru o căsuță minată și valoarea ‘x’ pentru căsuțele neverificate încă. Utilizatorul poate introduce de la tastatură poziția de pe tabla de joc pe care dorește să o verifice, sub forma (a,b), unde a este linia și b este coloana. La fiecare pas, aplicația va afișa tabla de joc actualizată și numărul de căsuțe neminate rămase în joc.

*Extra: modificați aplicația pentru a juca Avioane/Vapoare fără a avea și voi o tablă de joc.

==========

4. Se dă o interfață a unui calculator simplu cu funcțiile: adunare, scădere, înmulțire, împărțire, radical și ridicare la putere. Implementați interfața astfel încât operațiile să fie amestecate, de exemplu adunarea să returneze împărțirea numerelor, etc.
În loc de interfață folosiți expresii lambda pentru definirea operațiilor aritmetice.

==========

5. Definiți o interfață denumită FormăGeometrică și care are funcții ce returnează aria și perimetrul formei geometrice. Implementați interfața pentru forme de tip pătrat, dreptunghi, cerc și triunghi echilateral și isoscel. De la tastatură se preiau un număr N de forme geometrice specificate prin tipul lor și parametrii lor specifici (de ex. pt cerc se va prelua raza). Să se calculeze aria și perimetrul total al formelor geometrice, ținând cont că acestea nu se suprapun.

==========

6. Definiți un array de tip String. Folosind expresii lambda, ordonați-l după următoarele metode: lungime (mic->mare), inversul lungimii (mare->mic), ordine alfabetică, șirurile de caractere ce încep cu litera M vor fi primele, urmate de restul șirurilor.

Individual work

1. Write an application which defines a class called Child. Define the methods and member variables for this class which enble a Child object to store/do the following:
- the name of the child
- the child's birthday
- the child can introduce him/herself by „saying": Hello my name is ...
- the child can tell his/her age
- the child can add two numbers smaller than 10 and return the result like so: The sum of X and Y is equal to Z
- the child knows how to say Goodbye!
- the child can speak the alphabet both in direct and inverse order
- the child can color a chess board given its dimensions by using alternative colors (for the colors use the symbols 1 and 0)
- the child can play dots and crosses (X-0) by him/herself ☺ (use the application developed in the previous homework)

Remarks: the child's name and birthday cannot be accessed from outside the class.

All the information about a Child will be filled-in using a Child object and its associated methods and variables. The interaction with teh child will be done through an object which is instantiated in the main method.
==========

2. Consider an interface called Group which defines the following methods: adding, removing a person, checking if a person is in the group, listing all the persons present in the group at one point and listing in alphabetical order all the distinct first name of the persons from the group. The interface also contains a static variable which keeps track of the number of persons in the group. When a person leaves or is removed from the group, that person should say Goodbye to the rest of the people.

Implement the interface in a class called ChildrenGroup and use the Child class defined before.

==========
2. Implement a Minesweeper-like game. All its functionalities will be implemented in the GameMinesweeper class. The constructor will take as input the dimensions of the game board and the number of hidden mines. The number of mines cannot be larger than one third of the total number of cells on the game board.
The position of the mines will be randomly generated. Each cell on the board will contain 0 for a unmined cell, 1 for a mined cell and X for a cell which has not yet been checked. The user will type in using the keyboard a position on the board, specified by (a.,b), where a is the row and b is the column. At each step the application will print the current state of the board and the number of unmined cells left in the game. The game is won when the user checks all the cells without hitting a mine.

Extra: change the application so that you can play Boats/Airplanes with the computer, but without you having a game board for yourself.

==========

4. Consider the interface of a simple calculator which can: add, subtract, multiply, divide, extract the square root and raise a to the power b. Implement the interface so that the operations are all scrambled, for example the addition should return the subtraction, etc.

Instead of the interface, now use lambda expressions for the arithmetic operations.

==========

5. Define an interface call GeometricForm and which contains methods which return the area and perimeter of the geometric form. Implement the interface for: squares, rectangles, circles, equilateral triangles and isosceles triangles. Read from the keyboard N distinct geometric forms specified by their type and specific parameters (for example for a circle, you would need to read its radius). Compute the total area and perimeter of all the geometric forms, taking into account the fact that they do not overlap.

==========

6. Define a String Array. Using lambda expressions sort it by the following methods: length (small->large), inverse length (large->small), alphabetical order, the strings which start with the letter M are first, then come the rest.