Lucrare de laborator Nr. 1

Bazele Java

Obiective

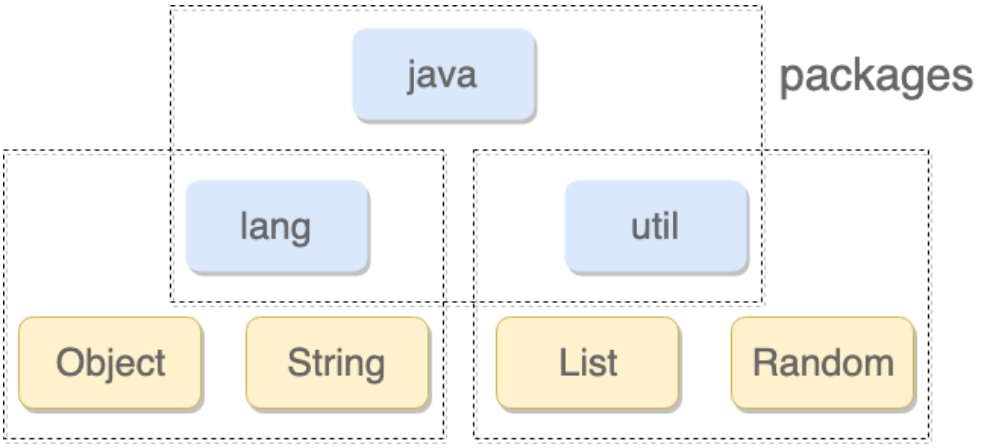
Scopul acestui laborator este familiarizarea studenților cu noțiunile de bază ale programării în Java.
Aspectele urmărite sunt:

- organizarea unui proiect Java
- familiarizarea cu IDE-ul
- definirea noțiunilor de clasă, câmpuri, proprietăți, metode, specificatori de acces
- folosirea unor tipuri de date

Organizarea unui proiect Java

În cadrul acestui laborator veți folosi IntelliJ IDEA. Primul pas este să vă familiarizați cu structura unui proiect și a unui fișier sursă Java.

Înainte de a începe orice implementare, trebuie să vă gândiți cum grupați logica întregului program pe unități. Elementele care se regăsesc în același grup trebuie să fie **conectate în mod logic**, pentru o ușoară implementare și înțelegere ulterioară a codului. În cazul Java, aceste grupuri logice se numesc **pachete** și se reflectă pe disc conform ierarhiei din cadrul proiectului. Pachetele pot conține atât alte pachete, cât și fișiere sursă.



Următorul pas este delimitarea entităților din cadrul unui grup, pe baza unor trăsături individuale. În cazul nostru, aceste entități vor fi clasele. Pentru a crea o clasă, trebuie mai întâi să creăm un fișier aparținând proiectului nostru și unui pachet (dacă este cazul și proiectul este prea simplu pentru a-l împărți în pachete). În cadrul acestui fișier definim una sau mai multe clase, conform următoarelor reguli:

1. dacă dorim ca această clasă să fie vizibilă din întreg proiectul, îi vom pune specificatorul **public** (vom vorbi despre specificatori de acces mai în detaliu în cele ce urmează); acest lucru implică însă 2 restricții:
   - fișierul și clasa publică trebuie să aibă același nume
   - nu poate exista o altă clasă/interfață publică în același fișier (vom vedea în laboratoarele următoare ce sunt interfețele)
2. pot exista mai multe clase în același fișier sursă, cu condiția ca **maxim una** să fie publică

Tipuri primitive

Conform POO, **orice este un obiect**, însă din motive de performanță, Java suportă și tipurile de bază, care nu sunt clase.

```
boolean isValid = true;
char nameInitial = 'L';
byte hexCode = (byte)0xdeadbeef;
short age = 23;
int credit = -1;
long userId = 169234;
float percentage = 0.42;
double money = 99999;
```

Mai multe informații despre tipuri primitive găsiți aici.

Biblioteca Java oferă clase wrapper (“ambalaj”) pentru fiecare tip primitiv. Avem astfel clasele Char, Integer, Float etc. Un exemplu de instanțiere este următorul:

```
new Integer(0);
```

Clase

Clasele reprezintă tipuri de date definite de utilizator sau deja existente în sistem (din **class library** – set de biblioteci dinamice oferite pentru a asigura portabilitatea, eliminând dependența de sistemul pe care rulează programul). O clasă poate conține:

- **membri** (variabile membru (**câmpuri**) și proprietăți, care definesc starea obiectului)
- **metode** (funcții membru, ce reprezintă operații asupra stării).

Prin instanțierea unei clase se înțelege crearea unui obiect care corespunde unui șablon definit. În cazul general, acest lucru se realizează prin intermediul cuvântului cheie **new**.

Procesul de inițializare implică: declarare, instanțiere și atribuire. Un exemplu de inițializare este următorul:

```
Integer myZero = new Integer(0);
```

Un alt exemplu de clasă predefinita este clasa **String**. Ea se poate instanția astfel (**nu** este necesară utilizarea **new**):

```
String s1, s2;

s1 = "My first string";
s2 = "My second string";
```

Aceasta este varianta preferată pentru instanțierea **string**-urilor. De remarcat că și varianta următoare este corectă, dar **ineficientă**, din motive ce vor fi explicate ulterior.

```
s = new String("str");
```

Câmpuri (membri)

Un câmp este un obiect având tipul unei clase sau o variabilă de tip primitiv. Dacă este un obiect atunci trebuie inițializat înainte de a fi folosit (folosind cuvântul cheie **new**).

```
class DataOnly {
   int i;
```

```
    float f;
    boolean b;
    String s;
}
```

Declarăm un obiect de tip DataOnly și îl inițializăm:

```
DataOnly d = new DataOnly();

// set the field i to the value 1
d.i = 1;

// use that value
System.out.println("Field i of the object d is " + d.i);
```

Observăm că pentru a utiliza un câmp/funcție membru dintr-o funcție care nu aparține clasei respective, folosim **sintaxa**:

```
classInstance.memberName
```

Proprietăți

O proprietate este un câmp (membru) căruia i se atașează două metode ce îi pot expune sau modifica starea. Aceste doua metode se numesc **getter** si **setter**.

```
class PropertiesExample {
    String myString;

    String getMyString() {
        return myString;
    }

    void setMyString(String s) {
        myString = s;
    }
}
```

Declarăm un obiect de tip **PropertiesExample** și îi inițializăm membrul **myString** astfel:

```
PropertiesExample pe = new PropertiesExample();

pe.setMyString("This is my string!");

System.out.println(pe.getMyString());
```

Metode (funcții membru)

Putem modifica programul anterior astfel:

```
String s1, s2;

s1 = "My first string";
s2 = "My second string";
```

```
System.out.println(s1.length());
System.out.println(s2.length());
```

Va fi afișată lungimea în caractere a șirului respectiv. Se observă că pentru a aplica o funcție a unui obiect, se folosește sintaxa:

```
classInstance.methodName(param1, param2, ..., paramN);
```

Funcțiile membru se declară asemănător cu funcțiile din C.

Specificatori de acces

În limbajul Java (și în majoritatea limbajelor de programare de tipul OOP), orice clasă, atribut sau metodă posedă un **specificator de acces**, al cărui rol este de a restricționa accesul la entitatea respectivă, din perspectiva altor clase. Există specificatorii:

Tip primitiv	Definitie
private	limitează accesul doar în cadrul clasei curente
default	accesul este permis doar în cadrul *pachetului* (package private)
protected	accesul este permis doar în cadrul pachetului si în clasele ce moștenesc clasa curentă
public	permite acces complet

Atenție, nu confundați specificatorul **default** (lipsa unui specificator explicit) cu **protected**.

Încapsularea se referă la acumularea atributelor și metodelor caracteristice unei anumite categorii de obiecte într-o clasă. Pe de altă parte, acest concept denotă și ascunderea informației de stare internă a unui obiect, reprezentată de atributele acestuia, alături de valorile aferente, și asigurarea modificării lor doar prin intermediul metodelor.

Utilizarea specificatorilor contribuie la realizarea ***încapsulării***.

Încapsularea conduce la izolarea modului de implementare a unei clase (atributele acesteia și cum sunt manipulate) de utilizarea acesteia. Utilizatorii unei clase pot conta pe funcționalitatea expusă de aceasta, **indiferent de implementarea ei internă** (chiar și dacă se poate modifica în timp).

Exemplu de implementare

Clasa **VeterinaryReport** este o versiune micșorată a clasei care permite unui veterinar să țină evidența animalelor tratate, pe categorii (câini/pisici):

```
public class VeterinaryReport {
    int dogs;
    int cats;

    public int getAnimalsCount() {
        return dogs + cats;
    }

    public void displayStatistics() {
        System.out.println("Total number of animals is " + getAnimalsCount());
```

```
    }
}
```

Clasa VeterinaryTest ne permite să testăm funcționalitatea oferită de clasa anterioară.

```
public class VeterinaryTest {
    public static void main(String[] args) {
        VeterinaryReport vr = new VeterinaryReport();

        vr.cats = 99;
        vr.dogs = 199;

        vr.displayStatistics();
        System.out.println("The class method says there are " + vr.getAnimalsCount() + "
animals");
    }
}
```

Observații:

- Dacă *nu inițializăm* valorile câmpurilor explicit, mașina virtuală va seta toate *referințele* (vom discuta mai mult despre ele în laboratorul următor) la null și *tipurile primitive* la 0 (pentru tipul boolean la false).
- În Java *fișierul trebuie să aibă numele clasei (publice) care e conținută în el*. Cel mai simplu și mai facil din punctul de vedere al organizării codului este de a avea fiecare clasă în propriul fișier. În cazul nostru, VeterinaryReport.java și VeterinaryTest.java.

Obiectele au fiecare propriul spațiu de date: fiecare câmp are o valoare separată pentru fiecare obiect creat. Codul următor arată această situație:

```
public class VeterinaryTest {
    public static void main(String[] args) {
        VeterinaryReport vr = new VeterinaryReport();
        VeterinaryReport vr2 = new VeterinaryReport();

        vr.cats = 99;
        vr.dogs = 199;
        vr2.dogs = 2;

        vr.displayStatistics();
        vr2.displayStatistics();

        System.out.println("The first class method says there are " + vr.getAnimalsCount() + "
animals");
        System.out.println("The second class method says there are " + vr2.getAnimalsCount() + "
animals");
    }
}
```

Se observă că:

- Deși câmpul dogs aparținând obiectului vr2 a fost actualizat la valoarea 2, câmpul dogs al obiectului vr1 a rămas la valoarea inițiala (199). **Fiecare obiect are spațiul lui pentru date!**
- System.out.println(…) este metoda utilizată pentru a afișa la consola standard de ieșire
- Operatorul '+' este utilizat pentru a concatena două șiruri

- Având în vedere cele două observații anterioare, observăm că noi am afișat cu succes șirul “The first/second class method says there are ” + vr.getAnimalsCount() + “ animals”, deși metoda getAnimalsCount() întoarce un întreg. Acest lucru este posibil deoarece se apelează implicit o metodă care convertește numărul întors de metodă în șir de caractere. Apelul acestei *metode implicite* atunci când chemăm System.out.println se întâmplă pentru orice obiect și primitivă, nu doar pentru întregi.

Având în vedere că au fost oferite exemple de cod în acest laborator, puteți observa că se respectă un anumit stil de a scrie codul in Java. Acest **coding style** face parte din informațiile transmise in cadrul acestei materii și trebuie să încercați să îl urmați încă din primele laboratoare, devenind un criteriu obligatoriu ulterior în corectarea temelor. Puteți găsi documentația oficială pe site-ul Oracle. Pentru început, încercați să urmați regulile de denumire: http://www.oracle.com/technetwork/java/codeconventions-135099.html

Arrays

Vectorii sunt utilizați pentru a stoca mai multe valori într-o singură variabilă. Un vector este de fapt o matrice (array) unidimensională.

Exemplu definire vector de String-uri cu valorile deja cunoscute

```
String[] cars = { "Volvo", "BMW", "Ford" };
```

Exemplu creare și populare vector cu valori de la 1 la 20

```
int[] intArray = new int[20];
for (int i = 0; i < intArray.length; i++) {
  intArray[i] = i + 1;
}
```

- Înainte să populăm vectorul, trebuie declarat (**int[] intArray**) și alocată memorie pentru 20 elemente de tip int (**new int[20]**).
- Pentru a accesa lungimea vectorului, folosim câmpul **length** păstrat în vector.
- Indexarea elementelor într-un array începe de la 0.

Nice to know

- Fiecare versiune de Java aduce și concepte noi, dar codul rămâne cross-compatible (cod scris pt Java 8 va compila si rula cu Java 12). Dintre lucrurile adaugate in versiuni recente avem declararea variabilelor locale folosind var, ceea ce face implicit inferarea tipului. Codul devine astfel mai ușor de urmărit.

```
var labString = "lab 1" // type String
```

- Când folosiți biblioteci third-party o să observați că pachetele au denumiri de forma *com.organizationname.libname*. Acesta este modul recomandat pentru a le denumi, vedeți mai multe detalii pe pagina oficială. Există situații în care acest stil este obligatoriu, de exemplu o aplicație Android nu poate fi publicată pe Play Store dacă numele pachetelor sale nu respectă aceasta convenție.

Summary

- Codul Java este scris în interiorul claselor, enum-urilor și interfețelor
- Un fișier de tip java poate conține mai multe clase
- Numele singurei clase publice dintr-un fișier trebuie să fie același cu numele fișierului (fără extensia *.java*)

- Fișierele sunt grupate în pachete
- În interiorul unei clase declarăm
  - variabile
  - metode
  - alte clase
- **Clasele și metodele ar trebui să aibă o singură responsabilitate. Evitați metodele lungi și clase cu mai mult de un rol. Nu puneți toată logica în metoda *main*. Metoda main ar trebui să conțină doar câteva linii cu instațieri si apeluri de metodele specializate.**
- În declarația claselor, câmpurilor, metodelor putem adăuga specificatori de acces sau diverse keywords pe care le vom studia în următoarele laboratoare
  - specificatori de access: **public**, **private**, **default**, **protected**
- Java are coding style-ul său și este importantă respectarea lui, altfel proiectele, mai ales cele cu mulți dezoltatori ar arăta haotic.
- Tipuri de date primitive: int, long, boolean, float, double, byte, char, short. Câmpurile primitive se inițializează automat la instanțierea unei clase cu valori default: e.g. 0 pentru int.
- Clasa String oferă multe metode pentru manipularea șirurilor de caractere.

Exerciții

1. Instalați:
   a) Java JDK nu mai puțin de 12
   b) IDE Intellij IDEA Community
   c) Verificați din linia de comandă versiunea de java:
      - **javac -version** - comanda *javac* este folosită pentru compilare
      - **java -version** - comanda *java* este folosită pentru rulare

Task 1 (3p)

1. Creați folosind IDE-ul un nou proiect și adăugați codul din secțiunea **Exemplu de implementare** (pp. 4-6). Rulați codul din IDE.
2. Mutați codul într-un pachet *task1* (sau alt nume pe care îl doriți să îl folosiți). Folosiți-vă de IDE, de exemplu **Refactor** → **Move** pentru **IntelliJ**. Observați ce s-a schimbat în fiecare fișier mutat.

Task 2 (5p)

Creați un pachet *task2* (sau alt nume pe care îl doriți să îl folosiți). În el creați clasele:
- **Student** cu proprietățile: *name* (String), *year* (Integer)
- **Course**
  - cu proprietățile: *title* (String), *description* (String), *students* (**array** de clase Student - exemplu arrays). Referitor la **array list** puteți să vă documentați aici.
  - cu metoda: *filterYear* care întoarce o listă de studenți care sunt într-un an dat ca parametru.
- Nu folosiți vreun modificator de acces pentru variabile (nu puneți nimic în fața lor în afară de tip)
- Test cu metoda **main**. La rulare, ca argument în linia de comandă se va da un integer reprezentând anul în care este un student. Informați-vă despre rulare în linia de comandă aici.
  1. creați un obiect *Course* și 3-4 obiecte *Student*. Populați obiectul Course.
  2. afișați toți studenții din anul dat ca parametru. **Hint:** folosiți **Arrays.toString**(listStudenti). In clasa Student folosiți IDE-ul pentru a genera metoda **toString** (pentru Intellij **Code→Generate**…)
  3. rulați atât din IDE (modificati run configuration) cât și din linie de comandă

- Opțional, în loc de arrays pentru *filterYear* puteți să folosiți și obiecte de tip List, e.g. ArrayList. Puteți să vă informați despre ArrayList aici.

Task 3 (1p)

1. Creați două obiecte *Student* cu aceleași date în ele. Afișați rezultatul folosirii metodei *equals()* între ele. Discutați cu cadrul didactic despre ce observați și pentru a vă explica mai multe despre această metodă.
   - documentație - https://www.ms.sapientia.ro/~manyi/teaching/oop/oop_romanian/curs16/curs16.html

Task 4 (1p)

1. Adăugați modificatorul de acces ***'private'*** tuturor variabilelor claselor **Student** și **Course** (e.g. *private String name;)*
2. Rezolvați erorile de compilare adăugând metode **getter** și **setter** acestor variabile.
   a. Ce ați făcut acum se numește *încapsulare (encapsulation)* și este unul din principiile de bază din programarea orientată pe obiecte. Prin această restricționare protejați accesarea și modificarea variabilelor.
      - *Hint*: pentru a vă eficientiza timpul, folosiți IDE-ul pentru a genera aceste metode
        - **Code → Generate… → Getters and Setters**

Exerciții

1. Numere complexe
   a. Creați un proiect nou cu numele ComplexNumber.
   b. Creați clasa ComplexNumber.java. Aceasta va avea două campuri: re si im, ambele de tip float.
2. Operații
   a. Creați clasa Operations.java. Această clasă va implementa operațiile de adunare și înmulțire pentru numere complexe. Definiți în clasă Operations câte o metodă pentru fiecare operație;
   b. Testați metodele implementate.
3. Biblioteca
   a. Creați un proiect nou cu numele Bookstore.
   b. Creați clasa Book.java cu următoarele atribute: title, author, publisher, pageCount.
   c. Creați clasa BookstoreTest.java, pentru a testa viitoarele funcționalități ale bibliotecii. Completați această clasă, așa cum considerați necesar. Apoi, creați un obiect de tip carte și setați atributele introducând date de la tastatură. Pentru această folosiți clasa Scanner:

```
Scanner s = new Scanner(System.in);
String title = s.nextLine();
```

4. Verificați ca numărul de pagini introdus să fie diferit de zero. Puteți consulta documentația claselor String și Integer.
5. Verificări cărți
   . Creați o clasă nouă, BookstoreCheck, ce va conține două metode, cu câte 2 parametri de tipul Book.
      - Prima metodă va verifica dacă o carte este în dublu exemplar, caz în care va întoarce adevărat.

- A doua metodă va verifica care carte este mai groasă decât altă, și va întoarce cartea mai groasă.

a. Testați aceste doua metode.

6. Formatare afișare
   - Puteți consulta documentația clasei String
   - Afișați informația despre o carte în felul următor:

```
BOOK_TITLE: [insert_book_title]
BOOK_AUTHOR: [insert_book_author]
BOOK_PUBLISHER: [insert_book_publisher]
```

Cu următoarele precizări:

- Titlul cărții va fi scris în întregime cu **majuscule**
- Numele editurii va fi scris în întregime cu **minuscule**
- Numele autorului rămânând **neschimbat**

Specificatori de acces

1. Implementaţi o clasă *MyArrayList*, care va reprezenta un vector de numere reale, cu posibilitatea redimensionării automate. Ea va fi definită în pachetul *arrays*. Clasa va conţine următoarele metode:
   - un constructor fără parametri, care creează intern un vector cu 10 elemente
   - un constructor cu un parametru de tipul întreg, care creează un vector de dimensiune egală cu parametrul primit
   - o metodă numită add(float value), care adaugă valoarea value la finalul vectorului. Dacă se depăşeşte capacitatea vectorului, acesta se va redimensiona la o capacitate dublă
     - *Atenţie!* Există o diferenţă între capacitatea vectorului (dimensiunea cu care a fost iniţializat) şi numărul de elemente memorate efectiv în el (care este cel mult capacitatea).
   - o metodă numită contains(float value) care returnează true dacă value există în cadrul vectorului
   - o metodă numită remove(int index) care elimină valoarea aflată în vector la poziţia specificată de index (numerotarea începând de la 0); se va da un mesaj dacă indexul este invalid
   - o metodă numită get(int index) care va returna elementul aflat în poziţia index
   - o metodă numită size() care returnează numarul de elemente din vector
   - o metodă declarată public String toString() care va returna o reprezentare a tuturor valorilor vectorului ca un şir de caractere
2. Scrieţi o clasă de *test* separată pentru clasa *MyArrayList*, populând lista cu 3 elemente şi verificând că valorile întoarse de metoda *get* corespund, într-adevăr, poziţiilor aferente din vectorul intern clasei. Ce condiţii trebuie îndeplinite pentru atingerea scopului propus?
3. Scrieţi un scenariu de utilizare a clasei *MyArrayList*, astfel:
   - iniţializând-o cu o capacitate de 5 de elemente iar apoi inserând 10 elemente aleatoare utilizând doar metoda *add*
   - cautând 5 valori aleatoare în vector
   - eliminând 5 valori aleatoare din vector
4. De multe ori este bine ca programarea unor aplicaţii mari să se faca modular, un modul fiind gestionat de o anumită clasă. De asemenea, în cadrul acestor aplicaţii, puteţi avea situaţii în care o metodă are nevoie să utilizeze mai multe module, deci să primească referinţele mai multor clase, fapt ce poate deveni dificil de gestionat. De aceea, aceste clase se pot implementa limitând accesul la *o singură instanţă statică*, ceea ce permite accesarea acesteia doar pe

baza *clasei*, fară a mai fi necesară trimiterea ei ca parametru in metode. Aşa descoperim un prim *design pattern* orientat-obiect; el se numeşte **Singleton** şi puteţi afla câteva despre el aici. Ne propunem să implementăm o variantă simplă. Creaţi clasa *Singleton*, care să

- aibă intern o singură instanţă *statică*
- aibă un constructor *privat* (de ce e nevoie?)
- expună o metodă *publică* de acces la instanţă (un getter)