

Clase și obiecte

1. Scopul lucrării

Obiectivele de învățare ale acestei sesiuni de laborator sunt cunoașterea și stăpânirea:

- Învățarea modului corect de declarare a claselor, definirea și instanțierea variabilelor de tip referință, apelul corect al metodelor
- Înțelegerea variabilelor Java și a modului de folosire a acestora în expresii
- Acumularea de experiență de proiectare și programare pentru cele de mai sus

2. Clase simple

2.1. Declararea claselor

Clasele sunt cărămizile de bază în construcția programelor Java. Clasele pot fi comparate cu planurile pentru clădiri. În loc să specifice structura clădirilor, clasele descriu structură "lucrurilor" dintr-un program. Aceste lucruri sunt apoi create ca obiecte software ale programului. Lucrurile care merită să fie reprezentate sub forma claselor sunt de obicei substantivele importante din domeniul problemei. De exemplu, o aplicație „cărucior pentru cumpărături bazat pe Web” e probabil să conțină clase care reprezintă clienți, produse, comenzi, linii de comenzi, cărți de credit, adrese pentru livrare și furnizori de produse.

Pentru declararea unei clase în Java folosiți următoarea sintaxă:

```
[public] [abstract|final] class NumeClasa [extends NumeClasaParinte] [implements NumeInterfete]
{
    // variabilele și metodele sunt declarate în interiorul acoladelor clasei
}
```

<input type="text"/>	Clauze import
<input type="text"/>	Comentariu pentru clasă Descriere a clasei în formatul pentru javadoc
class <input type="text"/> {	Numele clasei
<input type="text"/>	Declarații Datele partajate de mai multe metode se declară aici
<input type="text"/>	Metodă
...	
<input type="text"/>	Metodă
}	

- O clasă poate avea vizibilitate **public** sau *implicită* (fără modificator de acces).
- Poate fi sau **abstract**, **final** sau *concretă* (fără modificator).
- Trebuie să folosiți cuvântul cheie **class**, urmat de un identificador legal.
- Opțional clasa poate extinde una dintre clasele din ascendență. Implicit va extinde **java.lang.Object**.
- Opțional poate implementa oricâte interfețe, separate prin virgulă.
- Variabilele și metodele clasei se declară între acoladele exterioare '{}', care urmează după identificadorul clasei.
- Fiecare fișier sursă .java poate conține doar o singură clasă publică. Un fișier sursă poate conține orice număr de clase cu nivelul de acces implicit.
- Numele fișierului sursă trebuie să fie identic cu cel al clasei publice.

2.2. Constructori

La crearea unei noi instanțe a unei clase (un obiect nou), folosind cuvântul cheie **new**, este invocat un constructor pentru clasa respectivă. Constructorii sunt folosiți pentru a inițializa variabilele instanță (câmpurile) unui obiect. Constructorii sunt asemănători metodelor, dar există câteva diferențe importante. **Numele constructorului este numele clasei.** Un constructor trebuie să aibă același nume cu clasa în care se află.

Constructorul implicit. Dacă nu definiți un constructor pentru o clasă, compilatorul creează automat un implicit, fără parametri. Constructorul implicit invocă constructorul implicit pentru părinte și inițializează toate variabilele instanță la valorile implicite (zero pentru tipurile numerice, null pentru referințe la obiecte și false pentru booleene).

Constructorul implicit este creat numai atunci când nu sunt definiți constructori. Dacă definiți constructori pentru o clasă, atunci nu se mai creează automat un constructor implicit.

Diferențe între metode și constructori :

- Constructori nu au tip returnat. Valoarea este obiectul însuși, așa că nu este nevoie să se indice o valoare returnată.
- Nu există instrucțiune return în corpul constructorului.
- Prima linie din corpul constructorului trebuie să fie ori un apel la un alt constructor al aceleiași clase (folosind **this**), ori un apel al constructorului superclasei (folosind **super**). Dacă prima linie nu este nici unul dintre apeluri, compilatorul inserează automat un apel la constructorul fără parametri al superclasei.

Aceste diferențe de sintaxă dintre un constructor și o metodă sunt uneori greu de văzut în sursă. Poate ar fi fost mai bine să existe un cuvânt care să marcheze clar constructorii, așa cum sunt în unele limbaje.

this(...) – Apelează un alt constructor din aceeași clasă. Adesea un constructor cu mai puțini parametri apelează un constructor cu mai mulți parametri dând valori implicite parametrilor care nu sunt prezenți. Folosiți acest apel pentru constructori din aceeași clasă.

super(...) – Folosiți *super* pentru a apela un constructor dintr-o clasă părinte. Apelul constructorului pentru superclasă trebuie să fie prima instrucțiune din corpul unui constructor. Dacă constructorul implicit al superclasei satisface nevoile, atunci nu este nevoie să faceți apelul, deoarece acesta se va face automat. Super va fi folosit și exemplificat la capitolul despre moștenire.

Exemplu de apel explicit al constructorului this:

```
public class Point {
    int x;
    int y;

    //===== Constructor cu parametri
    public Point(int px, int py) {
        x = px;
        y = py;
    }

    //===== Constructor fara parametri
    public Point() {
        this(0, 0); // Apeleaza constructorul cu parametri; creează punctul (0, 0)
    }
    . . .
}
```

2.3. Declararea metodelor

O sintaxă generală pentru declararea metodelor este:

```
[modificatori] tip_returnat nume_metoda (lista_parametri) [clauza_throws]
{
    [lista_instrucțiuni]
}
```

Tot ce este între paranteze pătrate [] este opțional. Bineînțeles că nu scrieți parantezele pătrate în cod; aici ele sunt folosite pentru a indica elementele opționale. O declarație minimală de metodă cuprinde:

- **Modificatori:** set de cuvinte cheie ce definesc accesul la metode (**modificatori de acces**) sau anumite proprietăți speciale ale metodelor (**modificatori non-access**).
- **Tipul returnat:** tipul returnat este fie un tip Java valid (primitiv sau clasă) sau **void** dacă nu se returnează nici o valoare. Dacă metoda declară un tip returnat, atunci fiecare cale de ieșire din metodă trebuie să aibă o instrucțiune **return**.
- **Numele metodei:** numele metodei trebuie să fie un identificator Java valid.
- **Lista de parametri:** parantezele care urmează după numele metodei conțin zero sau mai multe perechi tip/identificator care constituie lista de parametri. Fiecare parametru este separat cu o virgulă. Lista de parametri poate fi vidă.
- **Throws:** o listă de excepții aruncate de metoda
- **Acoladele:** corpul metodei este cuprins între acolade. În mod normal corpul metodei conține o listă de instrucțiuni Java separate prin punct-și-virgulă care se execută secvențial. Tehnic, totuși, corpul metodei poate fi vid.

Numele metodei combinat cu lista de parametri pentru fiecare metodă dintr-o clasă trebuie să fie unic.

Unicitatea unei liste de parametri ia în considerare ordinea parametrilor.

Astfel că `int myMethod(int x, String y)` este diferită de `int myMethod(String y, int x)`.

2.3.1. Modificatorii de acces

Vizibilitatea unei metode (cunoscută și ca zona în care este accesibilă) definește ce obiecte o pot invoca și dacă subclasele o pot suprascrie. Cei patru modificatori de vizibilitate sunt: `public`, `protected`, `private`, și fără modificador. Păstrarea cât mai ascuns cu putință a metodelor unui obiect ajută la simplificarea API (Application Programming Interface: specificația care definește cum poate accesa programatorul metodele și variabilele unui set de clase). Nu faceți metoda mai vizibilă decât este necesar. Spre exemplu, dacă metoda urmează să fie suprascrisă într-o subclasă, dar nu va fi apelată niciodată de codul client, faceți vizibilitatea `protected`, nu `public`. Dacă o metodă nu trebuie niciodată invocată de o altă clasă și nu urmează să fie suprascrisă, faceți-o `private`.

Lista modificatorilor de acces este definită în următorul tabel:

Modificator	Poate fi accesată de
<code>public</code>	Orice clasă
<code>protected</code>	Clasa care o deține, orice subclasă, orice clasă din același pachet (package)
Fără modificador	Clasa care o deține, orice clasă din același pachet (package)
<code>private</code>	Clasa care o deține

2.3.2. Modificatorii non-access

Pe lângă modificatorii de acces, în plus, o metodă poate fi descrisă de următorul set de cuvinte cheie (denumit și set al modificatorilor non-access).

Cuvinte Cheie	Descriere
Vizibilitate	Poate fi una dintre valorile: <code>public</code> , <code>protected</code> , sau <code>private</code> . Determină care clase pot invoca metoda.
<code>static</code>	Metoda poate fi invocată la nivel de clasă, în loc de nivelul instanței clasei. Spre exemplu, <code>String.valueOf(35)</code> apelează <code>valueOf</code> pe clasa <code>String</code> în loc de un anumit obiect <code>String</code> . Desigur, metodele statice pot fi apelate și pe instanțe de clasă (obiecte): <code>myString.valueOf(35)</code> .
<code>abstract</code>	Metoda nu este implementată. Clasa trebuie extinsă și metoda trebuie implementată în subclasă.
<code>final</code>	Metoda nu poate fi suprascrisă într-o subclasă.
<code>native</code>	Metoda este implementată în alt limbaj.
<code>synchronized</code>	Metoda necesită să fie obținut un monitor (<i>lock</i>) de către codul care o invocă înainte de execuția metodei. Utilizat în cazul execuției <i>multi-thread</i> .

2.3.3. Parametri sunt transmiși prin valoare

În Java, transmiterea unui argument unui **apel de metodă** se face **prin valoare**.

La transmiterea unei valori primitive spre o metodă, se face o copie a valorii primitive. **Copia este de fapt manipulată de metodă.** Așa că, deși valoarea copiei poate fi schimbată în metodă, valoarea originală rămâne neschimbată.

La transmiterea unei **referințe spre un obiect** sau a unei **referințe spre un tablou** spre o metodă, **metoda manipulează de fapt o copie a acelei referințe.** Așa că, **metoda poate schimba atributele obiectului.** Dar, dacă re-asignează referința la un alt obiect sau un alt tablou, re-asignarea afectează doar copia, nu referința originală.

Un exemplu de transmitere a obiectelor este dat mai jos. Deși, aparent cele două metode *modify1* și *modify2* par să facă același lucru, rezultatele sunt diferite (discuții la problema 4.1).

```
public class Person {
    private String name;
    public Person(String name) { this.name = name; }
    public String getName() { return this.name; }
    public void setName(String newName) { this.name = newName; }
    public static void modify1(Person p) {
        p = new Person("Modified Name");
        System.out.print(p.getName());
    }
    public static void modify2(Person p) {
        p.setName("Modified Name");
        System.out.print(p.getName());
    }
    public static void main(String[] args) {
        Person p = new Person("Initial Person");
        modify1(p);
        System.out.println("Method modify1(Person p):" + p.getName());
        modify2(p);
        System.out.println("Method modify2(Person p):" + p.getName());
    }
}
```

2.3.4. Supraîncărcarea metodelor

Supraîncărcarea metodelor implică folosirea unui termen pentru a indica semnificații diverse. Supraîncărcarea unei metode Java înseamnă că scrieți mai multe metode cu același nume, dar cu argumente diferite.

Un exemplu de supraîncărcare a unei metode:

```
public int test(int i, int j) {  
    return i + j;  
}  
  
public int test(int i, byte j) {  
    return i + j;  
}
```

3. Variabile

Variabilele sunt locații din memorie în care se păstrează valori. Variabilele sunt de mai multe feluri și fiecare limbaj de programare abordează diferit caracteristicile acestora:

- **Numele variabilei.**
- **Tipul de dată** specifică natura informației pe care o variabilă o poate memora. Java are două tipuri generale de date.
 - 8 tipuri de bază sau *primitive* (byte, short, int, long, float, double, char, boolean).
 - Un număr nelimitat de tipuri *obiect* (String, Color, JButton, ...). O variabilă obiect Java păstrează o *referință* (pointer) către obiect, și nu obiectul propriu-zis, care este întotdeauna memorat în zona heap.
- **Domeniul** unei variabile stabilește cine poate să o vadă. Domeniul unei variabile este determinat în mare măsură de structura programului: d.e., bloc, metodă, clasă, pachet, subclasă.
- **Durata de viață** este intervalul de timp dintre momentul creării și respectiv al distrugerii unei variabile. Următoarele convenții sunt esențiale pentru înțelegerea funcționării sistemului Java. Variabilele locale și parametrii se creează la momentul inițierii unei metode și sunt distruse la momentul în care metoda execută return. Variabilele instanță sunt create de constructorul `new` și sunt distruse când nu mai există nici o referință către ele. Variabilele clasă (static) sunt create la momentul încărcării clasei și sunt distruse la terminarea programului.
- **Valori inițiale.** Ce valoare primește o variabilă atunci când este creată? Există mai multe posibilități.
 - Nici o valoare inițială. Variabilele **locale** Java **nu au valori inițiale**. Totuși compilatoarele Java fac o analiză simplă a fluxului de instrucțiuni pentru a se asigura de faptul că fiecărei variabile locale i se va atribui o valoare înainte de a fi utilizată. Mesajele de eroare privind variabilele neinițializate sunt de obicei justificate; uneori veți fi nevoiți să atribuiți o valoare inițială variabilei deși știți că acest lucru nu este neapărat necesar.
 - O valoare inițială specificată de utilizator. Java permite atribuirea de valori inițiale în enunțul de declarare a unei variabile.
 - Variabilele instanțe și statice au valori inițiale implicite: `zero` pentru numere, `null` pentru obiecte, `false` pentru variabile de tip boolean.
- **Obligativitatea declarațiilor.** Java, ca și alte limbaje de programare, vă pretinde să *declarați* variabilele – să informați compilatorul despre tipul variabilei etc. Declarațiile sunt importante deoarece ajută programatorul să construiască programe fiabile și eficiente.
 - Declarațiile permit compilatorului să găsească locurile în care o variabilă este utilizată eronat, d.e. parametri de tip eronat. Faptul că astfel de erori sunt găsite în timpul compilării ajută mult la reducerea timpului de testare a programelor. Anomaliile (bugs) nedepistate de compilator sunt mult mai greu de localizat și există riscul de a nu fi găsite decât după ce programul a fost livrat clientului.

- O declarație este de asemenea locul ideal pentru a scrie un comentariu ce descrie variabila respectivă și modul în care aceasta este utilizată.
- Deoarece declarațiile furnizează compilatorului mai multe informații, acesta poate genera cod mai bun.

Variabile locale/instanță/clasă

Limbajul Java prevede trei categorii de variabile:

- **Variabile locale** sunt declarate în interiorul unei metode, constructor sau bloc. La momentul introducerii unei metode, o zonă asociată metodei se plasează în *stiva de apeluri*. Această zonă conține câte o înregistrare pentru fiecare variabilă locală și fiecare parametru. Când este apelată metoda, fiecare înregistrare parametru este inițializată cu valorile parametrului respectiv. La momentul terminării metodei zona este eliminată din stivă și memoria devine disponibilă pentru următoarea metodă apelată. Parametrii sunt variabile locale esențiale care sunt inițializate cu valorile parametrilor actuali. Variabilele locale nu sunt vizibile în afara metodei.
- **Variabile instanță** sunt declarate într-o clasă, dar în afara unei metode. Ele mai sunt numite și *variabile membru* sau *variabile câmp*. Când un obiect este alocat în zona *heap*, se creează câte o înregistrare pentru fiecare valoare a variabilelor instanță. Astfel, o variabilă instanță este creată/distrusă odată cu obiectul căruia îi aparține. Variabila instanță este vizibilă în toate metodele și în toți constructorii care aparțin clasei care o definesc. În general trebuie declarată privată, dar i se poate conferi și o vizibilitate sporită.
- **Variabile clasă/statice** sunt declarate precizând cuvântul cheie `static` în interiorul unei clase, dar în afara unei metode. Există o singură copie per clasă, indiferent câte obiecte au fost create din ea. Ele sunt memorate în zona de memorie statică. În majoritatea cazurilor, variabilele statice se introduc în declarații `final` și se utilizează pe post de constante publice sau private.

Caracteristica	Variabile locale	Variabile instanță	Variabile clasă
Unde se declară	Metodă, constructor, sau bloc.	Într-o clasă, dar în afara unei metode. De obicei <code>private</code> .	Într-o clasă, dar în afara unei metode. Trebuie declarate <code>static</code> . De obicei au și atributul <code>final</code> .
Utilizare	Variabilele locale păstrează valori utilizate în calcule într-o metodă.	Variabilele instanță păstrează valori ce trebuie referite în mai mult de o metodă. (d.e. componente ce păstrează valori de genul șiruri de caractere, variabile pentru realizarea desenelor, etc.), sau care sunt părți esențiale ale stării unui obiect ce trebuie să existe între invocarea a două metode diferite.	Variabilele clasă sunt utilizate în general pentru constante, variabile care nu își modifică niciodată valorile lor inițiale.
Durata de viață	Sunt create la introducerea metodei sau a constructorului. Sunt distruse la ieșire.	Sunt create la momentul creării unei instanțe prin <code>new</code> . Sunt distruse când nu mai există nici o referire la obiectul care le conține (ele sunt preluate de colectorul de reziduuri « garbage collector »).	Sunt create la pornirea programului. Sunt distruse la terminarea programului.

Domeniu/ Vizibilitate	Variabilele locale (inclusiv parametrii formali) sunt vizibili numai în metoda, constructorul sau blocul unde au fost declarate. Modificatorii de acces (<code>private</code> , <code>public</code> , ...) nu pot fi utilizați pentru variabile locale. Toate variabilele locale sunt efectiv <code>private</code> pentru blocul în care au fost declarate. Ele nu sunt vizibile din nici o altă parte a programului, cu excepția metodei/blocului unde au fost declarate. Un caz special îl constituie o variabilă locală declarată în partea de inițializare a unei instrucțiuni <code>for</code> ; aceasta are drept domeniu domeniul instrucțiunii respective.	Variabilele instanță (field) sunt vizibile de către toate metodele unei clase. Numărul claselor pentru care mai sunt vizibile este determinat de atributul lor de acces. Alegerea implicită în declararea lor ar trebui să fie <code>private</code> . Nici o altă clasă nu poate vedea variabilele instanță <code>private</code> . Aceasta ar fi cea mai bună alegere. Pentru a păstra flexibilitatea reprezentării interne, și pentru a întări consistența datelor se recomandă a defini metode pentru citirea respectiv inițializarea variabilei dacă valorile trebuie aduse din afara clasei. Implicit (convenție numită și vizibilitatea pachetului) o variabilă poate fi văzută din orice clasă a aceluiași pachet. Cu toate acestea <code>private</code> este de preferat. <code>public</code> – poate fi văzută din orice clasă. În general este o idee greșită. <code>protected</code> – variabilele sunt vizibile numai din orice clasă de descendenți. Este o alegere ieșită din comun și probabil o alegere greșită.	La fel ca și variabilele instanță, dar deseori sunt declarate <code>public</code> pentru a oferi utilizatorilor clasei valorile constantelor.
Declarare	Declarația trebuie plasată într-o metodă sau bloc oriunde înainte de utilizare.	Oriunde la nivelul clasă (înainte sau după utilizare).	Oriunde la nivelul clasă împreună cu atributul <code>static</code> .
Valori inițiale	Niciuna. Trebuie să i se atribuie o valoare înainte de prima utilizare.	<i>Zero</i> pentru numere, <i>false</i> pentru variabile booleene, <i>null</i> pentru referințe la un obiect. Valorile pot fi atribuite într-o declarație sau într-un constructor.	La fel ca și variabilele instanță, dar în plus li se pot atribui valori într-un bloc special de inițializare statică.
Acces din afară	Imposibil. Numele variabilelor locale este cunoscut numai în interiorul metodei.	Variabilele instanță trebuie declarate <code>private</code> pentru a asigura ascunderea informației, astfel ele nu mai pot fi accesate din afara clasei. Totuși există situații mai rar întâlnite în care ele trebuie accesate din afara	Variabilele clasă sunt calificate de numele clasei (d.e., <code>Color.BLUE</code>). Ele pot fi calificate și de către un obiect dar este un stil amăgitor (poate crea confuzie) .

		clasei; în aceste cazuri variabilele vor fi calificate de un obiect (d.e. myPoint.x).	
Sintaxa numelor	Reguli standard	Reguli standard, dar deseori sunt prefixate pentru a face vizibilă diferența față de variabilele locale, d.e. my, sau m (pentru variabile membru) myLength, sau this ca și în numele this.length.	Variabilele (constantele) static public final se scriu toate cu majuscule, altfel ele se conformează convențiilor obișnuite de numire.

Accesarea variabilelor statice și apelul metodelor statice

Variabilele și metodele ne-stactice nu pot fi accesate/apelate din interiorul metodelor statice. Pentru a accesa/apela variabilele și metodele ne-stactice din interiorul unei metode statice, acest lucru se poate face doar dacă se creează un obiect prin intermediul căruia acestea pot fi accesate.

Câteva exemple:

- Metodele statice nu pot accesa variabile/metode ne-stactice

```
class Ex{
    int size = 42;
    void go() { }
    static void doMore() {
        int x = size;
        go();
    }
}
```

- Variabilele și metodele ne-stactice pot fi accesate doar de un obiect

```
class Ex{
    int size = 42;
    void go() { }
    static void doMore() {
        Ex f = new Ex();
        int x = f.size;
        f.go();
    }
}
```

- Metodele statice pot accesa doar variabile/metode care la rândul lor sunt statice

```
class Ex{
    static int count;
    static void woo() { }
    static void doMore() {
        woo();
        int x = count;
    }
}
```


Exemplu de variabile de clasă (statice)

```

class Dog {
    static int dogCount = 0;
    public Dog() {
        dogCount += 1;
    }
    public static void main(String[] args) {
        new Dog();
        new Dog();
        new Dog();
        System.out.println("Frog count is now " + dogCount);
    }
}

```

4. Mersul lucrării

4.1. Studiați și înțelegeți exemplele din laborator.

4.1.1. Parcurgeți codul din secțiunea 2.3.3.

- a) Fără a compila și executa acest cod, încercați să scrieți pe hârtie rezultatul afișat în linia de comandă în cazul apelării celor două metode *modify1* și *modify2*.
- b) Compilați și rulați codul dat ca exemplu. Comparați rezultatele obținute.

4.1.2. Ce mesaj ar fi afișat în ultimul exemplu cu variabile de clasă din secțiunea 3, dacă variabila *dogCount* ar fi una instanță și nu una statică?

4.2. Scrieți un program Java în care:

- Proiectați și implementați o clasă Autovehicul care să fie caracterizat prin: *marca*, *culoare* (vezi clasa predefinită Color), *viteza curentă* (în km/oră), *treapta de viteza curenta*, *viteza maximă pe care o poate atinge*. Un autovehicul ar trebui să poată efectua următoarele acțiuni: *accelerare* – care are ca efect creșterea vitezei cu un număr de km/oră, *decelerare*, *schimbarea treptelor de viteza*, *oprire*.
- Simulați deplasarea unui autovehicul pe o anumită distanță într-o aplicație statică. (Aceasta ar trebui să presupună acțiunile de accelerare/decelerare, schimbare de viteză, număr de km parcurși, oprire etc.)

Indicații de implementare:

- modificatorul de acces al variabilelor instanță să fie de tip private, iar cel al metodelor de tip public sau protected.
- Pentru accesul extern la variabilele instanță, implementați metode accesoare (*get*) și mutatoare (*set*) acolo unde este cazul.
- Definiți cel puțin doi constructori: cu și fără parametri.

4.3. Pornind de la programul dezvoltat la punctul anterior, adăugați în clasa Autovehicul următoarele:

- O constantă caracteristică pentru clasa Autovehicul
- Supraîncărcați o metodă

4.4. Adăugați clasei Autovehicul și un șofer. Definiți clasa Sofer care să fie caracterizată prin *nume*, *prenume*, *vârstă*, *număr permis de conducere*.

4.5. Extindeți programul vostru prin propunerea și integrarea a încă două clase care să aibă legătură cu clasele Autovehicul și/sau Sofer.

4.6. Proiectați și implementați un program nou - la alegere - care să conțină minim două clase și să simuleze activități/fenomene din lumea reală (ex.: extragere loto, împrumut de cărți de la bibliotecă, managementul angajaților la o firmă etc.).