FIȘA LABORATOR 2

Obiective

Lucru cu *obiecte*: definire, instanțiere, procesare, manipulare
Procesare = citire atribute sau proprietăți în structuri de „calcul”
Manipulare = modificarea valorilor atributelor/proprietăților

Concepte [Sub-teme țintă]

Creare structură clasă cu componente atribute - variabile de instanță
Completare clasă cu operații ale căror metode invocă variabilele de instanță
Completare clasă cu proprietăți (încapsulare atribute) – operații get/set
Instanțiere obiecte prin constructor implicit și inițializare explicită variabile de instanță
Invocare operații asupra obiectelor
Completare clasă cu metode constructor care inițializează variabile de instanță
Instanțiere prin constructor expliciți
Gestiune obiecte în tablouri
Parcurgere tablouri varianta clasică - for cu iterație pe index
Parcurgere tablouri varianta simplificată - for cu iterație pe variabilă
Gestiune obiecte în Liste
Parcurgere liste: *for* cu iterator-index și *for* cu iterator-referință

Desfășurare-Repere

Exemplu de predare
Enunț_1 problemă de clasa a III-a :
Un pix valoarea 10 lei și un caiet valorează 4,50 lei. Cât vor costa 2 pixuri și 3 caiete.

(i) Creare clasă *Produs* structurată prin atributele *denumire* și *pretUnitar;*

(ii) Codificare *soluția1()* în clasa de test *Main:*
- creare și inițializare obiecte simple;
- procesare prin access direct la variabilele de instanță (atributele) obiectelor

```
/*
 * Introducere creare obiecte
 */
static void solutia1(){
        // Creare obiecte simple
        Produs c = new Produs();
        // Initializare explicita obiecte (cu valori primitive)
        c.denumire = "Caiet";
        c.pretUnitar = 4.50;

        Produs p = new Produs();
```

```
        p.denumire = "Pix";
        p.pretUnitar = 10.0;

        // Absolut orice valoare cu care lucram o gestionam ca o variabila/printr-o structura de date
        Double cantitatePixuri = 2.0;
        // Procesare cu access valoare atribut obiect
        Double valoarePixuri = cantitatePixuri * p.pretUnitar;

        Double cantitateCaiete = 3.0;
        Double valoareCaiete = cantitateCaiete * c.pretUnitar;

        Double valoareTotala = valoarePixuri + valoareCaiete;

        String mesajRezultat = cantitatePixuri + " pixuri si " + cantitateCaiete + " caiete valoreaza:
" +
                valoareTotala + " lei";
        System.out.println(mesajRezultat);
}
```

(iii) Creare clasa *ArticolComanda* care va asocia *referințe* Produs la cantitate.

(iv) Însărcinare clasa *ArticolComanda* cu calcul valorii produselor vândute într-o anumită cantitate: adăugare metodă *calculValoare()* conținând implementarea algoritmului de calcul care face referire la variabile (interne) de instanță produs și cantitate.

(v) Codificare *soluția2()* în clasa de test *Main:*
- replicare *soluția1()*
- codificare instanțiere 2 obiecte de tip *ArticolComandă* și apelarea lor pentru obținerea valorilor produselor vândute.

```
/*
 * Introducere metode = incapsulare procesare ca responsabilitate in comportamentul unui obiect
 */
static void solutia2(){
        Produs c = new Produs();
        c.denumire = "Caiet";
        c.pretUnitar = 4.50;

        Produs p = new Produs();
        p.denumire = "Pix";
        p.pretUnitar = 10.0;

        // Obiect creat pentru asociere Produs si cantitate
        ArticolComanda pixuri = new ArticolComanda();
        // Initializare explicita valoare-referinta atribut
        pixuri.produs = p;
        pixuri.cantitate = 2.0;
        Double valoarePixuri = pixuri.calculValoare();

        ArticolComanda caiete = new ArticolComanda();
        caiete.produs = c;
        caiete.cantitate = 3.0;
        Double valoareCaiete = caiete.calculValoare();

        Double valoareTotala = valoarePixuri + valoareCaiete;

        String mesajRezultat = pixuri.cantitate + " pixuri si " + caiete.cantitate + " caiete
valoreaza: "
```

```
                + valoareTotala + " lei";
        System.out.println(mesajRezultat);
}
```

(vi) Introducere atribut intern privat *valoareArticol* în clasa *ArticolComanda* pentru stocarea rezultatului metodei *calculValoare*(): introducere metodă *geValoareComanda()* pentru access la valorile atributului în care va fi invocată metoda de calcul. Încapsulare accessului (get/set) este echivalentă cu expunerea atributelor indirect ca proprietăți (și nu direct prin variabile de inițializare accesibile public).

(vii) Încapsulare membri interni – atributele produs și cantitate prin generare proprietăți (Eclipse>Source\Generate getters and setters)

(viii) Codificare *soluția3()* în clasa de test *Main:*
- replicare *soluția2()*
- înlocuire access direct la atributele obiectelor (sintaxa varObiect.numeAtribute) cu accesul prin intermediul get/set;

De remarcat: creșterea lizibilității codului sursă, precum și o oarecare reducere a numărului de instrucțiuni.

```
/*
 * Introducere lucru cu proprietati
 */
static void solutia3(){
        Produs c = new Produs();
        /* Accesul la membri-atribute se face prin intermediul conventiei get/set*/
        c.setDenumire("Caiet");
        /* Sintaxa apelului metodelor get/set capata o semnatica mai explicita
         * decit simplele instructiuni de atribuire valori
         * catre variabile: getProprietate = obtine valoare proprietate
         * setProprietate = stabileste valoare proprietate
         * */
        c.setPretUnitar(4.50);

        Produs p = new Produs();
        p.setDenumire("Pix");
        p.setPretUnitar(10.0);

        // Obiect creat pentru asociere Produs si cantitate
        ArticolComanda pixuri = new ArticolComanda();
        // Initializare explicita valoare-referinta atribut
        pixuri.setProdus(p);
        pixuri.setCantitate(2.0);
        Double valoarePixuri = pixuri.getValoareArticol();

        /* Dpdv sematic apelurile orientate obiect sunt mult mai lizibile:
         * - creeaza un nou articol comanda
         * - pentru noul articol stabileste produsul c
         * - pentru noul articol stabileste cantitatea 3.0
         * - obtine valoarea noului articol
         */
        ArticolComanda caiete = new ArticolComanda();
        caiete.setProdus(c);
        caiete.setCantitate(3.0);
        Double valoareCaiete = caiete.getValoareArticol();

        Double valoareTotala = valoarePixuri + valoareCaiete;
```

```
        String mesajRezultat = pixuri.getCantitate() + " pixuri si " + caiete.getCantitate() + " caiete
                        valoreaza: " + valoareTotala + " lei";
        System.out.println(mesajRezultat);
}
```

(ix) Însărcinarea clasei *ArticolComanda* cu inițializarea explicită a membrilor interni produs și cantitate:
- generare constructor cu parametri pentru preluarea referinței produs și valorii pentru cantitate (Eclipse>Source\Generate constructor using parameters);
- generare constructor *default* – neparametrizat pentru rezolvarea erorilor de compilare apărute în unor inițializări din contexul *solutiilor 1, 2 si 3*.
(x) Adăugarea, în același mod, și a constructorilor (parametrizat și ne-parametrizat) în clasa *Produs*.

(xi) Codificare *soluția4()* în clasa de test *Main:*
- replicare *soluția3();*
- înlocuire inițializare directa la atributele obiectelor de tip Produs și ArticolComandă prin inițializare cu ajutorul constructorilor parametrizați;
De remarcat: creșterea lizibilității codului sursă, precum și o oarecare reducere a numărului de instrucțiuni.

```
/*
 * Introducere initializare prin constructor
 */
static void solutia4(){
        /*
         * Initializarea obiectelor se face mult mai elegant (mai putine linii de cod)
         * Numar de instructiuni se restringe semnificativ: de la 17 la 9
         */
        Produs c = new Produs("Caiet", 4.5);
        Produs p = new Produs("Pix", 10.0);

        ArticolComanda pixuri = new ArticolComanda(p, 2.0);
        Double valoarePixuri = pixuri.getValoareArticol();
        ArticolComanda caiete = new ArticolComanda(c, 3.0);
        Double valoareCaiete = caiete.getValoareArticol();
        Double valoareTotala = valoarePixuri + valoareCaiete;

        String mesajRezultat = pixuri.getCantitate() + " pixuri si " + caiete.getCantitate() + " caiete
                        valoreaza: " + valoareTotala + " lei";
        System.out.println(mesajRezultat);
}
```

(xii) Codificare *soluția5()* în clasa de test *Main* în scopul exemplificării gestionării obiectelor (referințelor obiectelor) în grupuri de obiecte - tablouri.
- replicare *soluția5();*
- adăugare instrucțiune inițializare tablou de obiecte;
- înlocuire inițializare variabile pentru articole cu inițializare elemente în tablou;
- adăugare structură de control repetitivă clasică (cu iterator index obiect) pentru parcurgerea tabloului în scopul cumulării valorilor;
- adăugare structură de control repetitivă generică (cu iterator-obiect) pentru parcurgerea tabloului în scopul formării mesajului rezultat;

```
/*
 * Introducere structuri tablou
```

```
 */
static void solutia5(){
        Produs c = new Produs("Caiet", 4.5);
        Produs p = new Produs("Pix", 10.0);

        /*
         * Cele doua articole nu or mai fi gestionate prin doua variabile ci
         * printr-un tablou cu doua elemente:
         * - declarare si initializare tablou
         * - initializare prim element tablou cu un nou articol
         * - initializare element secund tablou cu un nou articol
         * */
        ArticolComanda articole[] = new ArticolComanda[2];
        articole[0] = new ArticolComanda(p, 2.0);
        articole[1] = new ArticolComanda(c, 3.0);
        /*
         * Calculul valorii cumulate se face printr-o structurÄƒ repetitivÄƒ
         * de parcurgere a tabloului
         */
        Double valoareTotala = 0.0; // initializare valoare totala
        // Prima varianta parcurgere tablou: prin iterator numeric
        for(int i=0; i < articole.length; i++)
                valoareTotala = valoareTotala + articole[i].getValoareArticol();

        String mesajRezultat = "Rezultat: ";

        // A doua varianta parcurgere tablou: prin iterator-obiect
        for(ArticolComanda articol: articole)
                mesajRezultat += "cantitate: " + articol.getCantitate() +
                " - denumire: " + articol.getProdus().getDenumire() + " ... ";
        // Se remarca sintaxa cu punct repetitiva pentru a parcurge (naviga)
        // referintele obiectelor asociate

        mesajRezultat += " valoreaza: " + valoareTotala + " lei";
        System.out.println(mesajRezultat);
}
```

De remarcat: creșterea genericității soluției și simplificarea adusă de structură de control repetitivă cu iterator-obiect față de cea cu iterator-index.

(xiii) Codificare *soluția6()* în clasa de test *Main* în scopul exemplificării gestionării obiectelor (referințelor obiectelor) în grupuri de obiecte – colecții de tip listă. Considerentul de la care se pornește are în vedere asocierea grupului de articole cu valoarea totală calculată.
- adăugare clasa Comanda cu trei atribute (variabile de instanță) dataComanda, articole și valoareTotală. Toate atributele vor fi protejate, prin urmare declarate *private*. Atributul articole va trebui să gestioneze referințele tuturor articolelor comandate, prin urmare va fi tipizat ca o colecție *List* și inițializat prin clasa ArrayList. De asemenea, atributul *dataComanda* va fi inițializat cu data curentă.
- se generează metodele *get* și *set* care vor împacheta atributele și vor controla accesul la valorile interne ale acestora.
- se generează constructorul default;
- se generează metoda *calculValoareTotala()* pentru a încapsula în cadrul clasei Comanda algoritmul de calcul al valorii totale. Metoda *getValoareTotala()* va fi modificata pentru a se invoca *calculValoareTotala()* atunci când atributul intern *valoareTotala*. Se va reunuța la metoda *setValoareTotala()* pentru a elimina posibilitatea modificarii externe a valorii atributului, din moment ce aceasta rezultă doar din *calculValoareTotala()*.
- se adaugă metoda *adaugareArticol()* pentru a da posibilitatea lucrului cu valori-individuale ale colecției *articole*;

- se adaugă (suprascrie) metoda *de conversis toString()* în care se va codifica logica necesară detalieirii sub forma unui șir de caractere a *descriei textuale* a obiectelor de tp *Comanda*;
- replicare *soluția5();*
- adăugare instrucțiune de instanțiere a unei *Comenzi*;
- se codifică adaugarea elementelor din tabloul *articole* ca elemete ale proprietății-colecție cu același nume ale noi comenzi;
- se elimină logica de calcul a valorii totale și de construire a mesajului rezultat;
- se afișează detaliile comenzii prin preluarea acesteia ca argument în instrucțiunea de invocare a consolei de scriere.

```
/*
* Introducere structuri colectie de date ca proprietate a unui obiect
*/
static void solutia6(){
        Produs c = new Produs("Caiet", 4.5);
        Produs p = new Produs("Pix", 10.0);

        ArticolComanda articole[] = new ArticolComanda[2];
        articole[0] = new ArticolComanda(p, 2.0);
        articole[1] = new ArticolComanda(c, 3.0);

        Comanda comanda = new Comanda();
        comanda.adaugaArticol(articole[0]);
        comanda.adaugaArticol(articole[1]);

        System.out.println(comanda);
}
```

De remarcat: creșterea genericității și simplificarea elegantă a soluției

Exemplu lucru individual

Considerații finale

Clasele finale Produs, Comanda și ArticolComanda

```
package app;

public class Produs {
        String denumire;
        Double pretUnitar;

        /* Generare proprietati */
        public String getDenumire() {
                return denumire;
        }
        public void setDenumire(String denumire) {
                this.denumire = denumire;
        }
        public Double getPretUnitar() {
                return pretUnitar;
        }
        public void setPretUnitar(Double pretUnitar) {
                this.pretUnitar = pretUnitar;
        }

        /* Generare constructor */
        public Produs(String denumire, Double pretUnitar) {
                super();
                this.denumire = denumire;
```

```
                this.pretUnitar = pretUnitar;
        }
        public Produs() {
                super();
        }
}

/*-------------------------------------------------------------------------------------------------------------------*/
package app;

/* Clasa necesara pentru asociere ref-produse cu cantitate*/
public class ArticolComanda { // ArticolVanzare ???
        Produs produs;
        Double cantitate;

        /* Responsabilitate/Comportament articol: calcul valoare*/
        Double calculValoare(){
                Double valoare = null; // initializare variabila fara atribuire valoare-ref

                // access intern membri-variabile de instanta
                // valoare = produs.pretUnitar * cantitate;
                /* formula initiala se modifica usor pentru a permite calculul valorii
                 * in conditii valide: trebui sa existe un produs cu o cantitate
                 */
                if (produs != null && cantitate !=null)
                        valoare = produs.pretUnitar * cantitate;
                return valoare;
        }

        /* Formalizare proprietate valoare ????
         * set produs set cantitate => valoare noua
         * access valoare get: if null then calcul
         * */

        /* Variabila de instanta valoareArticol
         * - va stoca rezultatul calculValoare
         * - nu va fi accesibila direct (private):
         *              - e posibil ca sa nu fie initializata;
         * */
        private Double valoareArticol;
        public Double getValoareArticol(){
                if (valoareArticol == null)
                        valoareArticol = calculValoare();
                return valoareArticol;
        }

        /* - Metodele/Operatiile get/set transforma atributele produs si cantitate in proprietati
         * - Modificarea proprietatilor produs si cantitate tb sa produca automat actualizarea
         *   atributului valoareArticol accesibil ca si proprietate (get).
         * - Atributul valoareArticol nu are asociata si metoda set fiindca este o valoare
         *   calculata, adica nu va fi modificata direct din exterior
         * - Generare get/set pentru expunerea atributelor ca si proprietati:
         * Source/Generate getters and setters
         * */
        public Produs getProdus() {
                return produs;
        }
        public void setProdus(Produs produs) {
                this.produs = produs;
                valoareArticol = calculValoare();
        }
        public Double getCantitate() {
                return cantitate;
        }
```

```
	public void setCantitate(Double cantitate) {
		this.cantitate = cantitate;
		valoareArticol = calculValoare();
	}


	/*
	 * Incapsulare procedura de initializare in constructor
	 * Source/Generate constructor using fields
	 */
	public ArticolComanda(Produs produs, Double cantitate) {
		super();
		this.produs = produs;
		this.cantitate = cantitate;
	}


	/*
	 * Introducere explicita constructor default
	 * (rezolvare erori compilare pentru initializare explicita utilizata)
	 */
	public ArticolComanda() {
		super();
	}
}

/*-------------------------------------------------------------------------------------------------------------------------------*/
package app;
package app;

import java.util.ArrayList;
import java.util.Date;
import java.util.List;

public class Comanda {

	private Date dataComanda = new Date();
	private List<ArticolComanda> articole = new ArrayList<ArticolComanda>();

	/* Generare getteri si setteri */
	public Date getDataComanda() {
		return dataComanda;
	}
	public void setDataComanda(Date dataComanda) {
		this.dataComanda = dataComanda;
	}
	public List<ArticolComanda> getArticole() {
		return articole;
	}
	public void setArticole(List<ArticolComanda> articole) {
		this.articole = articole;
	}

	/* Generare constructor */
	public Comanda() {

	}


	/* Adaugare operatie manipulare detalii colectie*/
	public void adaugaArticol(ArticolComanda articol){
		articole.add(articol);
	}
```

```
        /* Adaugare proprietate valoare totala*/
        private Double valoareTotala;
        private void calculValoareTotala(){
                valoareTotala = 0.0;
                for (ArticolComanda articol: articole)
                        valoareTotala += articol.getValoareArticol();
        }
        public Double getValoareTotala(){
                if (valoareTotala == null)
                        calculValoareTotala();
                return valoareTotala;
        }

        /* Adaugare metoda info */
        public String toString(){
                String mesajRezultat = "articole: ";
                for(ArticolComanda articol: articole)
                        mesajRezultat += "cantitate: " + articol.getCantitate() + " - denumire: " +
                        articol.getProdus().getDenumire() + " ... ";
                mesajRezultat += " valoreaza: " + getValoareTotala() + " lei";

                return mesajRezultat;

        }
}
```