FIȘA LABORATOR 3

Obiective

Lucru cu *obiecte* complexe: model de lucru pentru *categorii structurale de obiecte* [1]
Procesare: parcurgere referințe înlănțuite.

Concepte [Sub-teme țintă]

Definire obiecte cu referințe – relații structurale [2]
Definire obiecte complexe – variantă implementare șablon compozit
Gestiune ***colecții*** (liste) structurale de obiecte (continuare)
Parcurgere recursivă relații obiecte în ierarhii
Supraîncărcare constructori
Compararea obiectelor: implementare operația ***equals*** – mecanism de identitate, verificare unicitate obiecte în colecții
Structuri repetitive ***for*** pentru colecții
Structuri repetitive ***recursive*** (apel recursiv al operațiilor) specifice șablonului „compozit”
Invocarea referinței obiectului curent prin ***this***

Desfășurare-Repere

Exemplu de predare
Un catalog prezintă o colecție de produse grupate în categorii și subcategorii. Un produs ar putea face parte din mai multe categorii, iar o categorie poate conține mai multe produse.

(a) Ce produse „acoperă” o categorie direct sau prin subcategorii ?
(b) Din ce categorii si subcategorii face parte un produs ?
(c) Cum s-ar putea lista produsele din catalog împreună cu categoriile din care fac parte ?

(i) Creați (dacă nu există deja) clasa **Produs** cu structura: idProdus, denumire, pretUnitar. „Decorați” clasa Produs după convențiile entity-bean:
- generați operațiile get/set pentru pentru fiecare variabilă de instanță, pentru a încapsula starea produselor (Source -> Generate getters and setters);
- generați 2 constructori: cel implicit (fărăr parametri) și un constructor cu parametri pentru toate variabilele de instanță (Source -> Generate constructor using parameters);
(ii) Creați clasa **Categorie** cu următoarea structură idCategorie, denumire, categorieParinte (folosind tipul Categorie), subCategorii (List-ă de Categorii – inițializare implicită prin ArrayList), produse (List-ă de

[1] Obiectele pot fi „împărțite” în categorii prin două mecanisme: definirea categoriilor drept obiecte ale unor clase specifice și asocierea acestor obiecte-categorii cu obiectele „categorisite” (categorii structurale) și definirea categoriilor drept clase specifice obiectele fiind „clasificate” prin instanțierea directă din clasele-categorii
[2] Obiectele pot face ***referire*** la alte obiecte sau gestiona referințe la alte obiecte prin: ***variabile de instanță*** (relații structurale) sau prin variabile sau parametri ***locali*** ai unor metode proprii.

Produse – inițializare implicită prin ArrayList). Generați getteri și setteri pentru toate variabilele de instanță și generați constructorul default (fără parametri) și un constructor cu parametri care să acopere membri non-colecții (idCategorie, denumire, categorieParinte). Completați clasa cu metodele ***adaugaProdus()*** și ***adaugaSubCategorie()*** pentru adaugarea individuală a elementelor în cele două colecții.

```
private List<Produs> produse = new ArrayList<Produs>();

public void adaugaProdus(Produs produs){
        this.produse.add(produs);
}
```

(iii) Creați clasa **Catalog**, cu următoarea structură: idCatalog, denumire și List-a de Categorii numită categoriiProduse cu inițializare din clasa ArrayList. Adăugați getterii, setterii, constructorul default și un constructor parametrizat pentru inițializare idCatalog și denumire. Completați clasa cu metoda ***adaugaCategorie()*** pentru gestiunea adăugării elementelor individuale în colecția categorii.

(iv) În clasa Main, metode principală ***main*** codul sursă necesar pentru:
- inițializare unui catalog cu numele Catalog 2011 având categoriile principale
- Software (cu subcategoriile Sisteme de operare și Office) și Hardware (cu subcategoriile Desktop și Mobile) și produsele
- MS Windows 7 și Apple iOS X făcând parte din subcategoria Sisteme de operare
- MS Office făcând parte din subcategoria Office
- PC Dell făcând parte din subcategoria Desktop
- NetBook Lenovo și iPhone 4S făcând parte din subcategoria Mobile.

```
public static void main(String[] args) {
        // 1 Catalog
        Catalog catalog = new Catalog(1, "Catalog 2011");
        // 2 Categorii principale din catalog
        Categorie categorie1 = new Categorie(1, "Software", null);
        catalog.adaugaCategorie(categorie1);
        Categorie categorie2 = new Categorie(2, "Hardware", null);
        catalog.adaugaCategorie(categorie2);
        // 4 Subcategorii
        Categorie categorie11 = new Categorie(11, "Sisteme de operare", categorie1);
        categorie1.adaugaSubCategorie(categorie11);
        Categorie categorie12 = new Categorie(12, "Office", categorie1);
        categorie1.adaugaSubCategorie(categorie12);
        Categorie categorie21 = new Categorie(21, "Desktop", categorie2);
        Categorie2.adaugaSubCategorie(categorie21);
        Categorie categorie22 = new Categorie(22, "Mobile", categorie2);
        Categorie2.adaugaSubCategorie(categorie22);
        // 6 Produse
        Produs p1 = new Produs(1, "MS Windows", 2150.0);
        categorie11.adaugaProdus(p1);
        Produs p2 = new Produs(2, "Apple iOS X", 2550.0);
        categorie11.adaugaProdus(p2);
        Produs p3 = new Produs(3, "MS Office", 850.0);
        categorie12.adaugaProdus(p3);
        Produs p4 = new Produs(4, "PC Dell", 1700.0);
        categorie21.adaugaProdus(p4);
        Produs p5 = new Produs(5, "NoteBook Lenovo", 2100.0);
        categorie22.adaugaProdus(p5);
        Produs p6 = new Produs(6, "iPhone 4S", 2400.0);
        categorie22.adaugaProdus(p6);
}
```

(v) **Afiseaza produse pornind de la o categorie**:
- adaugă în clasa Categorie responsabilitatea (operația) găsirii tuturor produselor din categoria respectivă ***parcurgând toate subcategoriile posibile*** prin metoda ***getProduseCategorie()*** care va returna o List-ă de produse. Se iau în calcul ***toate produsele asociate direct categoriei curente***, apoi se ***parcurg toate categoriile asociate de la care se cumulează (la rândul lor) toate produsele asociate direct sau recursiv prin subcategoriile asociate***.

```
public List<Produs> getProduseCategorie() {
        List<Produs> produseCategorie = new ArrayList<Produs>();
        // toate produsele asociate direct categoriei curente
        produseCategorie.addAll(this.produse);
        // se parcurg toate categoriile asociate
        for (Categorie categorie: this.subCategorii){
                // de la care se cumulează (la rândul lor) toate produsele asociate
                // direct sau recursiv prin subcategoriile asociate
                produseCategorie.addAll(categorie.getProduseCategorie());
        }
        return produseCategorie;
}
```

Testul, din clasa Main, ar arăta astfel:

```
System.out.println("-----------------------------------------------------");
System.out.println(categorie2.getDenumire() + " are urmatoarele produse:");
List<Produs> produseCategorie = categorie2.getProduseCategorie();
for (Produs p: produseCategorie)
        System.out.println("Produs categorie: " + p.getDenumire());
```

(vi) Ce se întâmplă dacă adăugăm de 2 ori produsul (referința) p6 (adică iPhone 4S) în categorie22 (Mobile), adică repetăm instrucțiunea `categorie22.adaugaProdus(p6);` ?
Rezultatul ar fi:

```
-----------------------------------------------------
Hardware are urmatoarele produse:
Produs categorie: PC Dell
Produs categorie: NoteBook Lenovo
Produs categorie: iPhone 4S
Produs categorie: iPhone 4S
```

Prin urmare respectivul produs se dublează. Pentru a putea "repara" această situație, metoda ***adaugareProdus*** ar putea fi modificată astfel:

```
private List<Produs> produse = new ArrayList<Produs>();

public void adaugaProdus(Produs produs){
        // adăugăm obiectul în colecție numai dacă nu este conținut deja de aceasta
        if (!this.produse.contains(produs))
                this.produse.add(produs);
}
```

(vii) Dar dacă, instanțiem din nou clasa Produs pentru un obiect cu același cod și nume

```
Produs p6_dublura = new Produs(6, "iPhone 4S - dublura", 2500.0);
categorie22.adaugaProdus(p6_dublura);
```

Rezultatul ar fi din nou o „dublare”:

```
--------------------------------------------------
Hardware are urmatoarele produse:
Produs categorie: PC Dell
Produs categorie: NoteBook Lenovo
Produs categorie: iPhone 4S
Produs categorie: iPhone 4S
Produs categorie: iPhone 4S - dublura
```

Prin urmare, deși produsele au același cod, colecția face diferența între ele comparându-le strict pe baza referinței lor interne. Pentru a modifica acest comportament și pentru a „compara” produsele după identificatorul lor logic (valoarea variabilei de instanță ***idProdus***) generăm metoda ***equals*** din Source -> Generate hashcode() and equals() bifând din lista de câmpuri doar ***idProdus***:

```
@Override
public int hashCode() {
        final int prime = 31;
        int result = 1;
        result = prime * result
                        + ((idProdus == null) ? 0 : idProdus.hashCode());
        return result;
}
@Override
public boolean equals(Object obj) {
        if (this == obj)
                return true;
        if (obj == null)
                return false;
        if (getClass() != obj.getClass())
                return false;
        Produs other = (Produs) obj;
        if (idProdus == null) {
                if (other.idProdus != null)
                        return false;
        } else if (!idProdus.equals(other.idProdus))
                return false;
        return true;
}
```

Putem proceda similar pentru ***idCatalog*** din clasa Catalog, și ***idCategorie*** din clasa Categorie.

(vii) **Afișează categoriile din care face parte un produs dintr-un catalog**:
- adaugă în clasa Catalog responsabilitatea (operația) parcurgerii tuturor categoriilor asociate catalogului care ar putea conține respectivul produs;

```
public List<Categorie> getCategoriiProdus(Produs produs){
        List<Categorie> categoriiProdus = new ArrayList<Categorie>();
        // ... parcurgerii tuturor categoriilor asociate catalogului
        for (Categorie categorie: categoriiProduse){
                // ia in calcul toate categoriile
                // ... care ar putea conține (apelul getSubCategoriiOf) respectivul produs
                categoriiProdus.addAll(categorie.getSubCategoriiOf(produs));
        }
        return categoriiProdus;
}
```

- adaugă în clasa Categorie responsabilitatea (operația) parcurgerii recursive a tuturor subcategoriilor care ar putea conține respectivul produs:

Varianta 1

```
public List<Categorie> getSubCategoriiOf(Produs produs){
        List<Categorie> categoriiProdus = new ArrayList<Categorie>();

        // Daca produsul cautat se afla intre produsele asociate direct
        // ia in cacul categoria curenta
        if (this.produse.contains(produs))
                categoriiProdus.add(this);

        List<Categorie> subCategoriiProdus = new ArrayList<Categorie>();
        // ... parcurgerii recursive (apelul getSubCategoriiOf) a tuturor subcategoriilor
        for (Categorie subCategorie: this.subCategorii){
                // ia in calcul toate categoriile
                subCategoriiProdus = subCategorie.getSubCategoriiOf(produs);
                // ... care ar putea conține respectivul produs
                categoriiProdus.addAll(subCategoriiProdus);
                // pentru efect cumulativ - adaugare subcategorii si supercategorii in calea de cautare
                if (!subCategoriiProdus.isEmpty())
                        categoriiProdus.add(this);
        }

        return categoriiProdus;
}
```

Varianta 2

```
public List<Categorie> getSubCategoriiOf(Produs produs) {
        List<Categorie> categoriiProdus = new ArrayList<Categorie>();

        // Daca produsul cautat se afla intre produsele asociate direct
        // ia in cacul categoria curenta
        if (getProduseCategorie().contains(produs))
                categoriiProdus.add(this);
        // ... parcurgerii recursive (apelul getSubCategoriiOf) a tuturor subcategoriilor
        for (Categorie subCategorie: subCategorii){
                // ia in calcul sub-sub categoriilor rezultate din apelul recursiv
                // adaugindu-le in rezultatul final ce va fi returnat
                categoriiProdus.addAll(subCategorie.getSubCategoriiOf(produs));
        }
        return categoriiProdus;
}
```

Testul, din clasa Main, ar arăta astfel:

```
System.out.println("-----------------------------------------------------");
System.out.println(p6.getDenumire() + "se gaseste in categoriile: ");
for (Categorie categorie: catalog.getCategoriiProdus(p6))
        System.out.println("->> " + categorie.getDenumire());
```

Exemplu lucru individual

Afișează produse din toate categoriile.

În clasa ***Catalog***

```
public List<Produs> getProduseCatalog(){
        List<Produs> produse = new ArrayList<Produs>();
        for (Categorie categorie: categoriiProduse){
                produse.addAll(categorie.getProduseCategorie());
        }
        return produse;
}
```

În ***main***

```
System.out.println("-----------------------------------------------------");
List<Produs> produseCatalog = catalog.getProduseCatalog();
String denCategoriiProdus;
for(Produs p: produseCatalog){
        denCategoriiProdus = "";
        for (Categorie c: catalog.getCategoriiProdus(p))
                denCategoriiProdus += c.getDenumire() + "-";
        System.out.println("Produs catalog: " + p.getDenumire() + " [" + denCategoriiProdus + "]");
}
```

Considerații finale

```
package app;

public class Produs {
        private Integer idProdus;
        private String denumire;
        private Double pretUnitar;

        public Integer getIdProdus() {
                return idProdus;
        }
        public void setIdProdus(Integer idProdus) {
                this.idProdus = idProdus;
        }
        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;
        }
        public Produs(Integer idProdus, String denumire, Double pretUnitar) {
                this.idProdus = idProdus;
                this.denumire = denumire;
                this.pretUnitar = pretUnitar;
        }
        public Produs() {
        }
```

```
	@Override
	public int hashCode() {
		final int prime = 31;
		int result = 1;
		result = prime * result
				+ ((idProdus == null) ? 0 : idProdus.hashCode());
		return result;
	}
	@Override
	public boolean equals(Object obj) {
		if (this == obj)
			return true;
		if (obj == null)
			return false;
		if (getClass() != obj.getClass())
			return false;
		Produs other = (Produs) obj;
		if (idProdus == null) {
			if (other.idProdus != null)
				return false;
		} else if (!idProdus.equals(other.idProdus))
			return false;
		return true;
	}

}

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

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

public class Categorie {
	private Integer idCategorie;
	private String denumire;

	private List<Categorie> subCategorii = new ArrayList<Categorie>();
	private Categorie categorieParinte;

	private List<Produs> produse = new ArrayList<Produs>();

	/*-----------------------------------------------*/
	public Integer getIdCategorie() {
		return idCategorie;
	}

	public void setIdCategorie(Integer idCategorie) {
		this.idCategorie = idCategorie;
	}

	public String getDenumire() {
		return denumire;
	}

	public void setDenumire(String descriere) {
		this.denumire = descriere;
	}

	public List<Produs> getProduse() {
		return produse;
	}

	public void setProduse(List<Produs> produse) {
```

```
		this.produse = produse;
	}

	public void adaugaProdus(Produs produs) {
		// if (!this.produse.contains(produs))
		this.produse.add(produs);
	}

	public Categorie getCategorieParinte() {
		return categorieParinte;
	}

	public void setCategorieParinte(Categorie categorieParinte) {
		this.categorieParinte = categorieParinte;
	}

	public Categorie(Integer idCategorie, String denumire,
			Categorie categorieParinte) {
		this.idCategorie = idCategorie;
		this.denumire = denumire;
		this.categorieParinte = categorieParinte;
		/* Relatii
		if (categorieParinte != null) {
			this.categorieParinte = categorieParinte;
			this.categorieParinte.adaugaSubCategorie(this);
		}*/
	}

	public Categorie() {
	}

	public List<Categorie> getSubCategorii() {
		return subCategorii;
	}

	public void setSubCategorii(List<Categorie> subCategorii) {
		this.subCategorii = subCategorii;
	}

	public void adaugaSubCategorie(Categorie categorie) {
		subCategorii.add(categorie);
	}

	/*-------------------------------------------------------------*/
	public List<Produs> getProduseCategorie() {
		List<Produs> produseCategorie = new ArrayList<Produs>();
		// toate produsele asociate direct categoriei curente
		produseCategorie.addAll(this.produse);
		// se parcurg toate categoriile asociate
		for (Categorie categorie : this.subCategorii) {
			// de la care se cumulează (la rândul lor) toate produsele asociate
			// direct sau recursiv prin subcategoriile asociate
			produseCategorie.addAll(categorie.getProduseCategorie());
		}
		return produseCategorie;
	}

	public List<Categorie> getSubCategoriiOf(Produs produs) {
		List<Categorie> categoriiProdus = new ArrayList<Categorie>();

		// Daca produsul cautat se afla intre produsele
		// categoriei curente
		if (this.produse.contains(produs))
			categoriiProdus.add(this);
```

```
                // ... parcurgerii recursive a tuturor subcategoriilor
                List<Categorie> subCategoriiProdus = new ArrayList<Categorie>();
                for (Categorie subCategorie : this.subCategorii) {
                        // ia in calcul toate categoriile
                        subCategoriiProdus = subCategorie.getSubCategoriiOf(produs);
                        // ... care ar putea conține respectivul produs
                        categoriiProdus.addAll(subCategoriiProdus);
                        // pentru efect cumulativ - adaugare subcategorii si supercategorii
                        // in calea de cautare
                        if (!subCategoriiProdus.isEmpty())
                                categoriiProdus.add(this);
                }

                return categoriiProdus;
        }

}

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

package app;

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

public class Catalog {
        private Integer idCatalog;
        private String denumire;
        private List<Categorie> categoriiProduse = new ArrayList<Categorie>();

        public Integer getIdCatalog() {
                return idCatalog;
        }

        public void setIdCatalog(Integer idCatalog) {
                this.idCatalog = idCatalog;
        }

        public String getDenumire() {
                return denumire;
        }

        public void setDenumire(String denumire) {
                this.denumire = denumire;
        }

        public List<Categorie> getCategoriiProduse() {
                return categoriiProduse;
        }

        public void setCategoriiProduse(List<Categorie> categoriiProduse) {
                this.categoriiProduse = categoriiProduse;
        }

        public Catalog(Integer idCatalog, String denumire) {
                this.idCatalog = idCatalog;
                this.denumire = denumire;
        }

        public Catalog() {
        }
```

```
	public void adaugaCategorie(Categorie categorie) {
		this.categoriiProduse.add(categorie);
	}

	/*------------------*/
	public List<Produs> getProduseCatalog() {
		List<Produs> produse = new ArrayList<Produs>();
		for (Categorie categorie : categoriiProduse) {
			produse.addAll(categorie.getProduseCategorie());
		}
		return produse;
	}

	public List<Categorie> getCategoriiProdus(Produs produs) {
		List<Categorie> categoriiProdus = new ArrayList<Categorie>();
		// parcurgerii tuturor categoriilor asociate catalogului
		for (Categorie categorie : categoriiProduse) {
			// ia in calcul toate categoriile
			// care ar putea conține respectivul produs
			categoriiProdus.addAll(categorie.getSubCategoriiOf(produs));
		}
		return categoriiProdus;
	}
}
```