FIȘA LABORATOR 4

Obiective

Polimorfism structural: model de lucru pentru *categorii-subclase de obiecte*[1]
Procesare: Test parcurgere polimorfică instanțe subclase.

Concepte [Sub-teme țintă]

Definire subclase: *moștenire* proprietăți.
Access proprietăți protejate [*protected*] în subclase.
Constructori în subclase și access constructori în super-clase
Subtipizare: operații de *up-casting* și *down-casting*
Clase *abstracte*: neinstanțiabile
Comparabilitate și *ordonare* obiecte: Object.equals(), Comparable.compareTo().
Colecții ordonate *TreeSet*.

Desfășurare-Repere

Exemplu de predare
Un catalog prezintă un colecție de produse grupate în clase-categorii cu subclase-subcategorii: Software (cu SistemeOperare și SuiteOffice) și Hardware (cu SistemeDesktop și SistemMobile). Parcurgeți catalogul de produse și afisați caracteristicile fiecărui produs în parte.

(i) Creați (dacă nu există deja) clasa **Produs** cu structura *idProdus*, *denumire*, *pretUnitar*. „Decorați" clasa Produs după convențiile entity-bean (vezi L3).

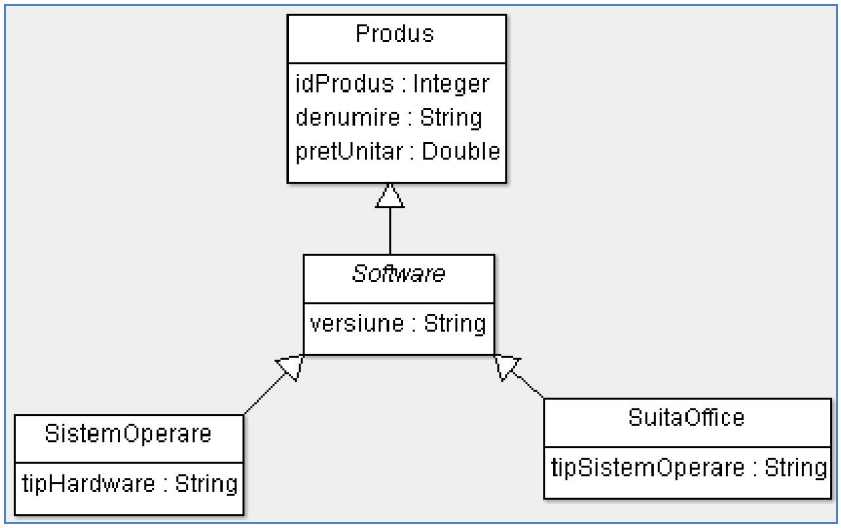
(ii) Creați subclasa ***Software***: în fereastra de definire a noii clase (din Eclipse: New-Class) selectați clasa *Produs* în rubrica *SuperClass*. Adăugați în clasa Software atributul *versiune* de tip String și metodele *get/set* aferente. Adăugați doi constructori (din Eclipse: Source – Generate constructor using fields):
- un constructor cu parametri: selectați în rubrica pentru *super constructorul invocat* (adică *Select super constructor to invoke*) constructorul *Produs cu parametri* pentru *idProdus*, *denumire*, *pretunitar*, la care adăugati din lista de câmpuri *versiune*;
- un constructor fără parametri: selectați în rubrica pentru *super constructorul invocat* constructorul *Produs() fără parametri*.

(ii) Creați subclasa ***SistemOperare***: în fereastra de definire a noii clase (New-Class) selectați clasa *Software* în rubrica *SuperClass*. Adăugați în clasa SistemOperare atributul *tipHardware* de tip String și metodele get/set aferente. Adăugați doi un constructori:

[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

- un constructor cu parametri: selectați în rubrica pentru *super constructorul invocat* constructorul *Software cu parametri* pentru *idProdus*, *denumire*, *pretunitar*, *versiune* la care adăugați din lista de câmpuri *tipHardware*;
- un constructor fără parametri: selectați în rubrica pentru *super constructorul* invocat constructorul *Software() fără parametri*.

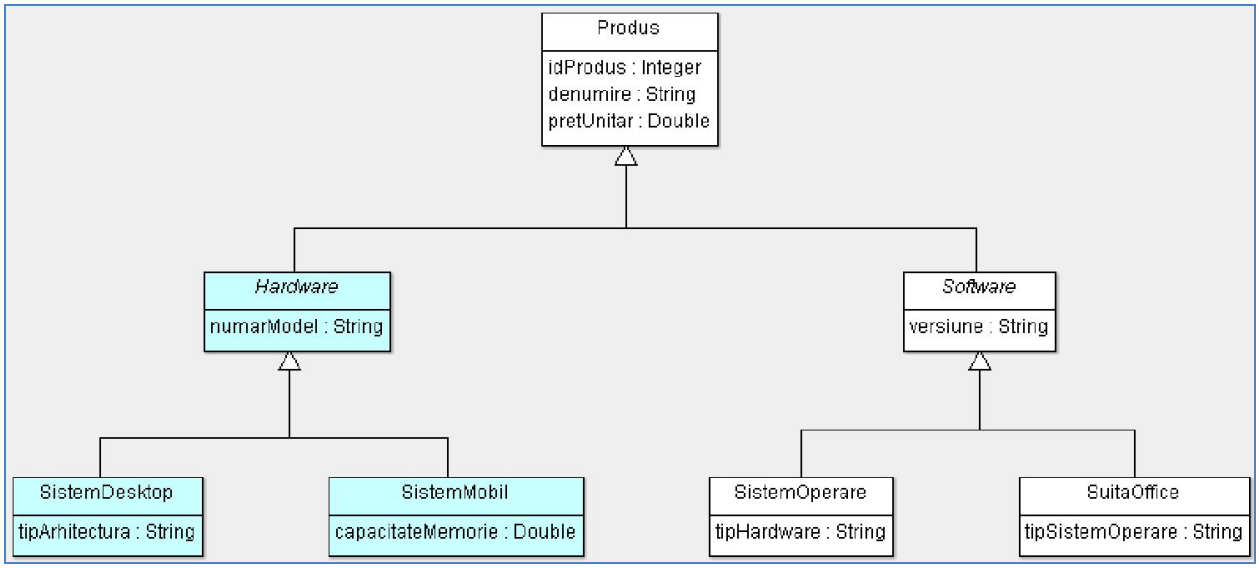
(iii) Creați în mod similar cu subclasa SistemOperare, subclasa **SuitaOffice** cu atributul *tipSistemOperare* de tip String




```
public abstract class Software extends Produs {
        protected String versiune;
        // get si set pentru versiune
        /*... ... ...*/
        // constructor parametrizat
        public Software(Integer idProdus, String denumire, Double pretUnitar,
                        String versiune) {
        /*... ... ...*/
        }
        // constructor fara parametri
        public Software() {
        }
}
/*************************************************************************************************************
*/
public class SistemOperare extends Software {
        private String tipHardware;
        // get si set pentru tipHardware
        /*... ... ...*/
        // constructor parametrizat
        public SistemOperare(Integer idProdus, String denumire, Double pretUnitar,
                        String versiune, String tipHardware) {
                /*... ... ...*/
        }
        // constructor fara parametri
        public SistemOperare(){
        }
}
/*************************************************************************************************************
*/
```

```
public class SuitaOffice extends Software {
        private String tipSistemOperare;
        // get si set pentru tipSistemOperare
        /*.. ... ...*/
        // constructor parametrizat
        public SuitaOffice(Integer idProdus, String denumire, Double pretUnitar,
                        String versiune, String tipSistemOperare) {
                /*... ... ...*/
        }
        // constructor fara parametri
        public SuitaOffice() {
        }
}
```

(iv) Creați, în mod similar cu ierarhia de mai sus, subclasele *Hardware*, *SistemDesktop* și *SistemMobil* care să completeze descrierea categoriilor de produse:



(v) Declarați clasa *Hardware* ca și clasă ne-instanțiabilă asociindu-i clasificatorul *abstract* (vezi clasa Software).

(vi) Creați o clasă de test (care să conțină metoda *main*) în care să instanțiați următoarele produse:
- SistemOperare(1, "MS Windows", 2450.0, "v7", "x86");
- SistemOperare(2, "Apple iOS", 1250.0, "v5", "iPhone");
- SuitaOffice(3, "MS Office", 850.0, "v2010", "Ms Windows 7");
- SistemDesktop(4, "PC Dell", 1700.0, "D100", "mini tower");
- SistemMobil(5, "NB Lenovo", 2100.0, "N5001", 450.0);
- SistemMobil(6, "iPhone", 3500.0, "4S", 32.0).

```
public class Test {
        public static void main(String[] args) {
                List<Produs> catalogProduse = new ArrayList<Produs>();
                SistemOperare p1 = new SistemOperare(1, "MS Windows", 2450.0, "v7", "x86");
                catalogProduse.add(p1);
                /* ... ... ... */
        }

}
```

(vii) Observați cuvântul *super* folosit în subclase și *probați* funcționalitatea acestuia.

(viii) Parcurgeți lista de produse și afișați proprietățile acestora funcție de *clasa din care sunt instanțiați* în mod concret.

```
public class Test {
	public static void main(String[] args) {

		List<Produs> catalogProduse = new ArrayList<Produs>();
		SistemOperare p1 = new SistemOperare(1, "MS Windows", 2450.0, "v7", "x86");
		// Up-casting
		catalogProduse.add(p1);
		/* ... ... ... */

		for(Produs p: catalogProduse){
			System.out.print(p.getDenumire());
			if(p.getClass().equals(SistemOperare.class)){
				// Down-casting
				SistemOperare so = (SistemOperare) p;
				System.out.println(so.getVersiune() +
					" ruleza pe tip hardware" + so.getTipHardware());
			}
			if(p.getClass().equals(SuitaOffice.class)){
				/* ... ... ... */
			}
			if(p.getClass().equals(SistemDesktop.class)){
				/* ... ... ... */
			}
			if(p.getClass().equals(SistemMobil.class)){
				/* ... ... ... */
			}
		}
	}

}
```

(ix) Comentați polimorfismul colecției *catalogProduse* manifestat:
- la adăugarea produselor concrete în catalog rezultat, polimorfism datorat ca efect al unei operații de up-casting al instanțelor concrete (SistemOperare, SuitaOffice etc.) la tipul de bază *Produs*;
- la parcurgerea produselor din colecție (prin blocul dedicat *for*) polimorfism datorat ca efect al unei operații de down-casting de la tipul de bază *Produs* la tipul claselor concrete (SistemOperare, SuitaOffice etc.).