TEHNICI AVANSATE DE PROGRAMARE
LABORATORUL 2

## 1. Declararea claselor:

```
[public][abstract][final] class <NumeClasa> [extends <NumeSuperclasa>]
[implements Interfata1 [, Interfata2 ... ]]
{
      // Corpul clasei
}
```

Implicit, o clasă poate fi folosită doar de clasele aflate în același pachet (librărie) cu clasa respectivă (dacă nu se specifică un anume pachet, toate clasele din directorul curent sunt considerate a fi în același pachet). O clasă declarată public poate fi folosită din orice altă clasă din care este vizibilă (care are acces la pachetul din care face parte).

**EXE:** Se dau clasele P1, C1 din pachetul pachet. Creați o clasă a altui pachet care să tipărească un obiect de tip C1. (folosiți import pachet)

```
package pachet;
public class P1 {
      int i;
      public C1 c1;
      P1(C1 c1){
            this.c1 = c1;
            i = c1.c;
      }

      public P1(int i){
            c1 = new C1(i);
            i = c1.c;
      }
}

class C1{
      int c;
      P1 p1;
      C1(int c)
      {
            this();
            this.c = ++c;
      }

      C1(){
            p1 = new P1(this);
      }

      public String toString(){
            return "c = " + c + " p = " + p1.i;
      }
}
```

Este corect apelul `P1 p = new P1()`?

## 2. Clase abstracte

O clasa este abstractă dacă este incomplet definită (conține cel puțin o metodă neimplementată). Doar clasele abstracte pot avea metode abstracte. Clasele abstracte nu pot fi instanțiate.
Dacă se doreste obținerea unei clase care nu poate fi instanțiată acest lucru se poate realiza și prin declararea constructorilor ca fiind de tip `private`.
Prin declararea clasei ca fiind de tip `final` se previne derivarea clasei.

**EXP:** Pentru a se defini o clasa abstractă `ClasaAbstracta` care conține metoda abstractă `metoda` se va folosi:

```
abstract class ClasaAbstracta {
	abstract void metoda();
}
```

**EXE:** Să se simuleze aruncarea unei monezi de mai multe ori. Folosind clasa `MasuraTimp` să se afle timpul necesar (în milisecunde) pentru apariția de 10 ori consecutiv a fetei 0 a unei monezi.
Aruncarea se simulează în modul următor:
Se generează un număr aleator a cuprins între 0 si 1.
a < 0.5 semnifică apariția fetei 0.
a > 0.5 semnifică apariția fetei 1.
Pentru generarea de numere aleatoare se va folosi un obiect al clasei `java.util.Random`

```
abstract class MasuraTimp {
	abstract void metoda();
	public long timp() {
		long t0 = System.currentTimeMillis();
		metoda();
		return System.currentTimeMillis()-t0;
	}
}
```

## 3. Extinderea claselor

Superclasa unei clase este precizată în clauza opțională `extends`. O clasă poate moșteni o singură clasă. Dacă lipseste clauza opțională `extends` atunci clasa Object este superclasa directă. Toate clasele derivă direct sau indirect din clasa Object. Object este singura clasa care nu are superclase.

**EXP:**

```
class Automobil {
	int nrKm = 1000;
	String culoare;

	int getCapacitate(){
		return 4;
	}

	Automobil(int nrKm, String culoare){
		this.nrKm += nrKm;
		this.culoare = culoare;
	}
}
```

```
class Duster extends Automobil{
	int pret;

	int getCapacitate(){
		return 5 ;
	}

	Duster(int nrKm, String culoare, int pret){
		super(nrKm, culoare);
		this.pret = pret;
	}

	public String toString(){
		return this.getClass() + " " + culoare + " in valoare de " +
		pret +" nr km = " + nrKm;
	}
}

public class TestAutomobil{
	public static void main(String args[]){
		Automobil auto = new Duster(2000,"alb",11000);
		System.out.print(auto);
	}
}
```

**EXE** Modificați clasa `TestAutomobil` astfel încât prețul care va fi trimis ca parametu constructorului clasei Duster să fie citit de la tastatură.
Creati o noua clasa `EroareValoare` care să extindă clasa `Exception`. Clasa `EroareValoare` va avea un constructor fară parametrii care va apela constructorul superclasei (`super('valoare incorecta')`).
Constructorul `Duster(int nrKm, String culoare, int pret)` va arunca o eroare de tip `EroareValoare` daca prețul este negativ. (`throw new EroareValoare();`)
Includeti apelul `Duster(int nrKm, String culoare, int pret)` într-un bloc `try catch`.

**OBS** Excepțiile care au ca superclasă clasa RuntimeException sau clasa Error nu trebuie tratate neapărat într-un bloc `try catch`. Este obligatorie tratarea erorilor din clasa Exception.

## 4. Polimorfism

Fie A o clasă si fie B o subclasă a sa si declarația:

```
A a = new B();
```

Vom spune că obiectul a are tipul declarat A si tipul real B, ceea ce pune în evidență noțiunea de polimorfism. Tipul real al unui obiect poate coincide cu tipul său declarat sau este o subclasă a tipului declarat.
Fie var un câmp al clasei A, ce este redeclarat (ascuns) în subclasa B. Dacă obiectul a face referire la câmpul var, atunci este vorba de câmpul declarat în clasa A, adică este folosit tipul declarat al obiectului.
Fie met o metodă a clasei A, care este rescrisă în subclasa B. La invocarea metodei met de către obiectul a, este folosită fie implementarea corespunzătoare metodei ascunse (dacă este statică), fie cea corespunzătoare metodei redefinite (dacă este nestatică). Cu alte cuvinte, pentru metode statice este folosit tipul declarat (la fel ca pentru câmpuri), iar pentru metode nestatice este folosit tipul real al obiectului.

**EXP:** A se vedea exemple – TestPolimorfism.java

**EXE:** Ce va returna apelul auto.getCapacitate() (consultați exemplul de la punctul anterior)

## 5. Interfețe

O interfață grupează un set de metode abstracte. Metodele unei interfețe sunt implicit abstracte (nu trebuie să fie precedate de cuvântul abstract).
Pe lângă metode abstracte interfețele pot conține declarațiile unor constante. Acestea pot fi sau nu declarate cu modificatorii public, static ,si final care sunt impliciți, nici un alt modificator neputând apărea în declarația unei variabile dintr-o interfață. Constantele unei interfețe trebuie obligatoriu inițializate.
O interfață poate moșteni o altă interfață. O clasă poate implementa mai multe interfețe, acesta fiind procedeul de realizare a mostenirii multiple.

**EXP:** A se vedea exemple – TestInterfete.java

## 6. Clasa Object

Clasa Object este superclasa directă a oricărei clase ce nu extinde altă clasă. Astefel Object este superclasă indirectă pentru orice alta clasă.
Metodele clasei Object sunt: toString(), clone(), equals(), finalize(), getClass(), hashCode().

**EXP:** Pentru realizarea clonării obiectelor, Java pune la dispoziție interfața Clonable și metoda clone a clasei Object. Intefața Clonable din pachetul java.lang va fi implementată de clasa în care va fi invocată clone. Metoda clone verifică dacă obiectul curent implementează Cloneable. În caz afirmativ este întors un obiect nou („clonă”).

```
class Obiect implements Cloneable{
	int continut;
	Obiect(int x){
		continut = x;
	}
	public Obiect clone(){
		try{
			return (Obiect) super.clone();
		}
		catch (CloneNotSupportedException e){
			return this;
		}
	}
	public String toString(){
		return " " + continut;
	}
}

public class Test{
	public static void main(String args[]){
		Obiect o1 = new Obiect(1);
		Obiect o2 = o1.clone();
		System.out.println(o1.equals(o2));
		System.out.println(o1 + " " + o2);
	}
}
```

**EXE 1:** Să se suprascrie în clasa Student din laboratorul 1 metoda equals astfel încât aceasta să returneze true dacă numele și prenumele coincid. Testați.

**EXE 2:** Să se scrie un program care listează fișierele cu extensia java dintr-un director dat. (Folosiți în exemplul de mai jos interfața java.io.FileFilter

```
public static void main(String[] args){
      File dir=new File(".");
      File[] fisiere=dir.listFiles();
      for (int i=0;i<fisiere.length;i++){
            System.out.println("Denumire:"+fisiere[i].getName()+
            "Dimensiune (bytes):" +fisiere[i].length());
      }
}
```