



**Examen POO – 5 februarie 2023**

1p din oficiu

1. (0,5p) Definiți conceptul de **polimorfism** în Programarea Orientată pe Obiecte. Dați un exemplu simplu în JAVA.
2. (0,5p) Dați un exemplu în JAVA de o **proprietate specifică fiecărui obiect instanță a unei clase**, care nu poate fi accesată direct decât din clasele aflate în același pachet cu clasa din care fac parte obiectele respective.
3. (0,5p) Definiți conceptul de **supraîncărcare a metodelor** în Programarea Orientată pe Obiecte. Dați un exemplu simplu în JAVA.
4. (0,5p) Explicați ce se întâmplă în urma compilării și execuției următorului program JAVA:

```
abstract class A {  
    public void info () {  
        System.out.println("Examen greu, ");  
    }  
}  
  
class B extends A{  
    public void info () {  
        super.info();  
        System.out.println("5 februarie 2023.");  
    }  
    public static void main(String[] args) {  
        B obj = new A();  
        obj.info();  
    }  
}
```

5. (0,5p) Dați un exemplu simplu în JAVA de **interfață cu o metodă care are comportament implicit definit**.
6. (0,5p) Explicați ce se întâmplă în urma compilării și execuției următorului program JAVA:

```
class A {  
    public int info() throws Exception{  
        return 5/0;  
    }  
}  
  
class B extends A {  
    public int info() {  
        return 10;  
    }  
    public static void main(String[] args) throws Exception{  
        A obj = new B();  
        System.out.println("Nota: "+ obj.info());  
    }  
}
```

7. (0,5p) Dați un exemplu simplu în JAVA de o **funcționalitate care nu poate fi executată simultan** de către două fire de lucru (*threads*) diferite.

8. (6x0,25p=1,5p) Scrieți pe 6 rânduri separate rezultatele afișate la ieșirea standard obținute în urma execuției următorului program JAVA:

```
class A {
    static String z="POO";
    String t = new String("Examen");
    int x=50;

    public A() {
        z+="A";
        x+=1;
    }

    public float method(float y) {
        return 5+y;
    }
}
```

```
class B extends A {
    int x=30;
    public B() {
        this.z+="B";
        super.x*=2;
    }

    public double method(double y) {
        return 5*y;
    }

    public String info() {
        String r="";
        int a[] = {1,2,3};
        try { a[3]=10; }
        catch (NullPointerException x) { r="M"; }
        catch (Error x) { r="P"; }
        catch (Exception x) { r="C"; }
        catch (Throwable x) { r="B"; }
        finally { r+="ERE"; }
        return r;
    }

    public static void main(String[] args) {
        A a = new A();
        A b = new B();
        System.out.println(b.x);
        System.out.println(b.method(2));
        A t[] = new A[3];
        System.out.println(A.z);
        System.out.println(b instanceof A);
        System.out.println(a.t==b.t);
        System.out.println(((B)b).info());
    }
}
```

9. (4p) Un sistem de management al rezultatelor competițiilor sportive stochează informații despre meciurile de fotbal și baschet. Pentru orice meci se reține data în care a avut loc, cele două echipe care au jucat, precum și rezultatul final sub forma unui scor (punctele realizate de fiecare echipă). În cazul unui meci de fotbal se mai reține și scorul la pauză (meciul având două reprize), iar în cazul unui meci de baschet scorul din fiecare sfert (meciul având patru sferturi). Fiecare echipă este caracterizată prin nume și anul înființării.

Respectând paradigma programării orientată pe obiecte, identificați entitățile, relațiile dintre acestea, desenați diagrama de clase UML detaliată (0,5p) și implementați în JAVA un program care să rezolve următoarele cerințe:

- (1p) Crearea de echipe prin citirea tuturor informațiilor de la intrarea standard și stocarea lor unică, în cadrul sistemului, într-o colecție permanent ordonată alfabetic după nume
- (0,75p) Crearea de meciuri specificând informațiile necesare și stocarea lor în cadrul sistemului. În cazul în care unul din scoruri (final sau de pe parcursul meciului) nu este valid (are valori numere negative), meciul nu va fi creat și se va arunca o excepție verificată.
- (0,5p) Determinarea echipelor de fotbal care au câștigat cele mai multe meciuri, prin analiza rezultatelor tuturor meciurilor existente în sistem
- (0,75p) Afișarea tuturor detaliilor despre toate meciurile din sistem, în ordinea cronologică a datei. Folosiți clasa existentă `Date`, cu constructorul `Date(int an, int luna, int zi)`, și care implementează interfața `Comparable` și metoda `compareTo` pentru compararea cronologică a două date calendaristice
- (0,5p) Scrieți o clasă cu o metodă *main* în care creați mai întâi sistemul de management al rezultatelor, apoi executați toate cerințele specificate mai sus astfel încât să utilizați fiecare funcționalitate cel puțin o dată