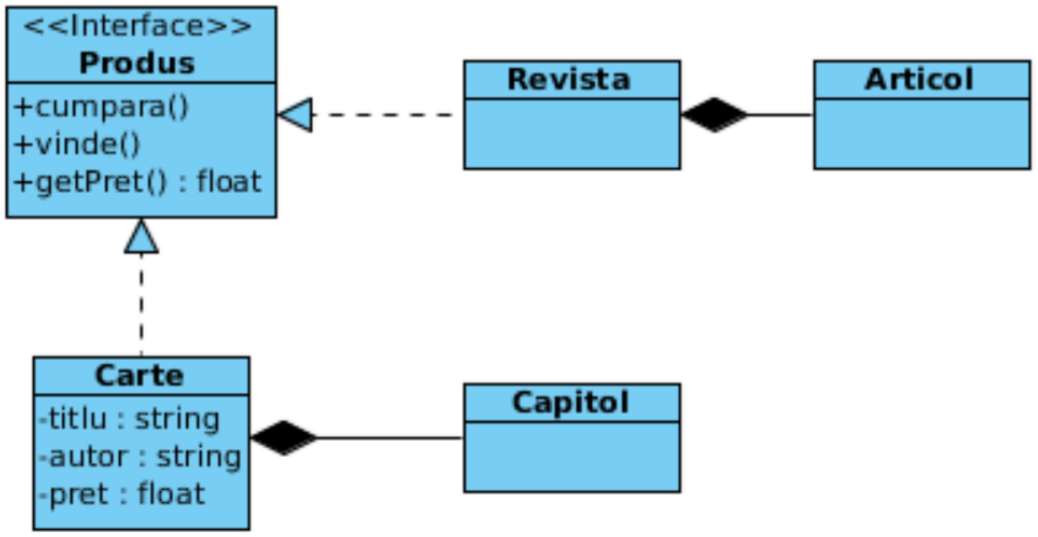
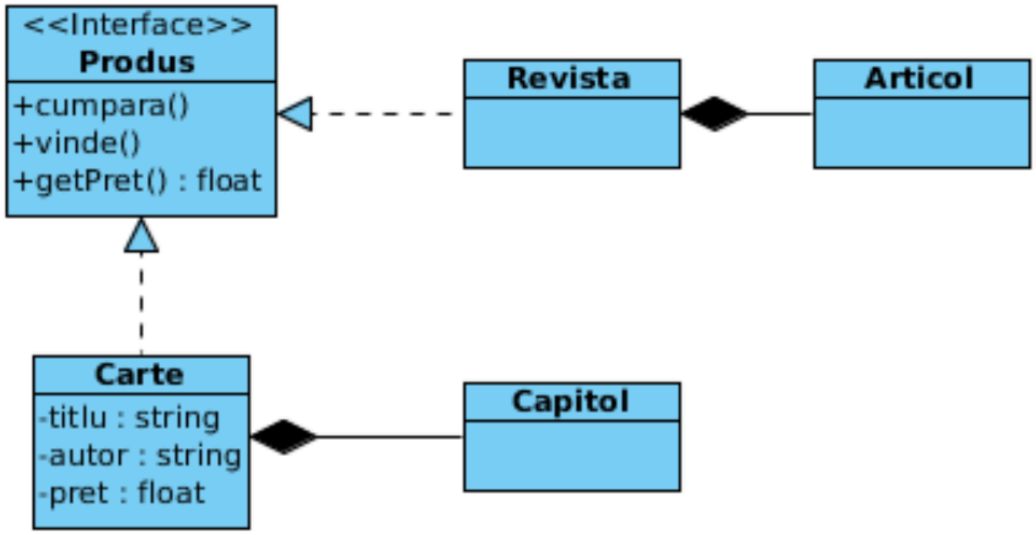
Tema 6

E1. Fie următoarea diagramă de clase.



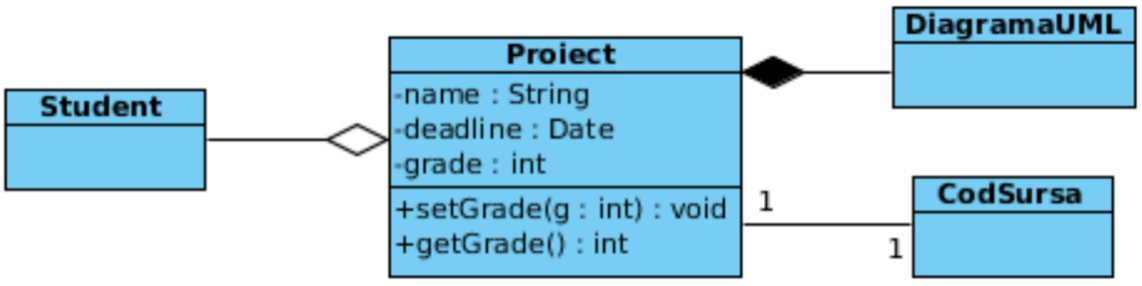
Selectați descrierea corectă și completă a relațiilor clasei Carte:
❌(a) Generalizare între clasa Carte(subclasă) și clasa Produs(superclasă); clasa Carte definește o compoziție de obiecte de tip Capitol.
[Relația între Produs și Carte este realizare, iar Produs este o interfață.]
❌(b) Generalizare între clasa Carte(subclasă) și interfața Produs(superclasă); clasa Carte definește o compoziție de obiecte de tip Capitol.
[Relația între Produs și Carte este realizare, nu de generalizare.]
❌(c) Realizare între clasa Carte și clasa Produs; clasa Carte implementează clasa Produs; clasa Carte definește o compoziție de obiecte de tip Capitol.
[Produs este o interfață, nu o clasă.]
✅(d) Realizare între clasa Carte și interfața Produs; clasa Carte implementează interfața Produs; clasa Carte e o compoziție de obiecte de tip Capitol.
❌(e) Realizare între clasa Carte și interfața Produs; clasa Carte implementează interfața Produs; clasa Capitol definește o compoziție de obiecte de tip Carte.
[Clasa Carte e o compoziție de obiecte de tip Capitol, nu invers.]

E2. Fie următoarea diagramă de clase.



Ce secvențe valide de cod Java rezultă din diagramă pentru clasa Revista?
❌(a) class Revista extends Produs{...}
[Produs nu e clasă, ci interfață. Nu e extends, ci implements.]
❌(b) class Produs implements Revista{...}
[Revista implements Produs, și nu invers.]
✅(c) private Collection<Articol> capitole = new Collection();
[Relația dintre Articol și Revista este de compoziție. Prin urmare, Revista are ca atribut o colecție/listă/tablou de articole.]
❌(d) public float getPret();
[Metoda este doar declarată, dar nu este implementată.]
❌(e) protected float getPret();
[Vizibilitatea metodei este public, ci nu protected.]
✅(f) class Revista implements Produs{...}
[Produs este interfață, iar clasa Revista o implementează.]
❌(g) public Produs vinde(){...}
[Return type-ul pentru metoda vinde nu este menționat și deci nu poate fi Produs.]

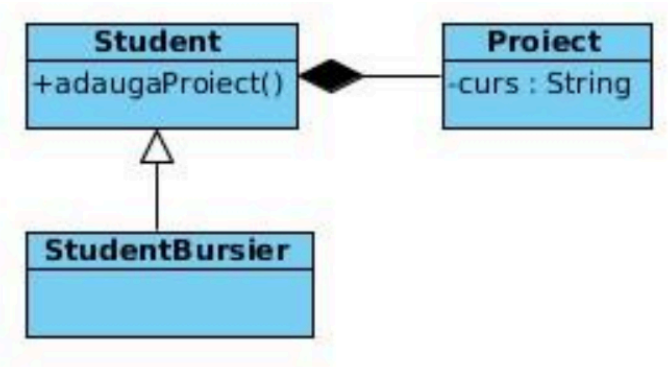
E3. Fie următoarea diagramă de clase.



Care secvențe de cod Java sunt valide pentru clasa Proiect?
✅(a) private CodSursa theCode;

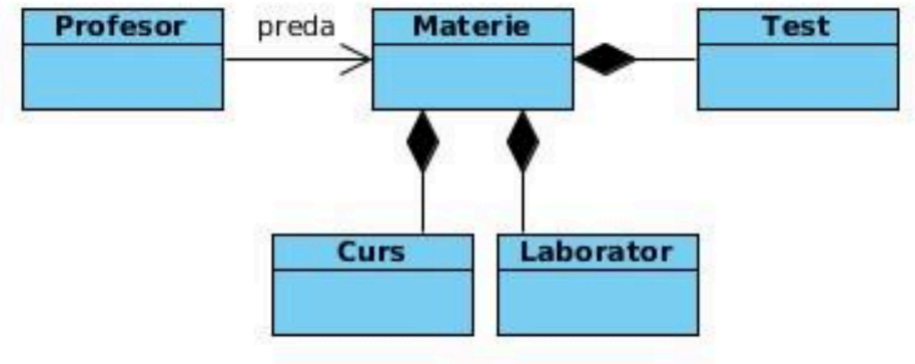
[CodSursa si Proiect sunt în relație de asociere: unui singur Proiect i se asociază un CodSursa.]
❌(b) private grade int;
[Sintaxa nu este corectă.]
✅(c) private Collection<DiagramaUML> diagrams = new Collection();
[Clasa Proiect si clasa DiagramaUML sunt în relație de compoziție. Proiect are ca atribut o colecție/listă/tablou de diagrame UML.]
❌(d) public Date deadline;
[Vizibilitatea atributului este private, ci nu public.]
❌(e) public getGrade(){...}
[SIntaxa nu este corectă. Nu este indicat return type-ul.]
❌(f) class Proiect extends Student{...}
[Relația dintre Proiect și Student nu este de moștenire, ci de agregare.]
✅(g) public void setGrade(int g){...}
[Metoda este declarată și implemetată corect.]
❌(h) private DiagramaUML diagram;
[Clasa Proiect poate conține mai multe diagrame UML. Nu doar un obiect de tip diagramă UML.]

E4. Fie următoarea diagramă de clase.



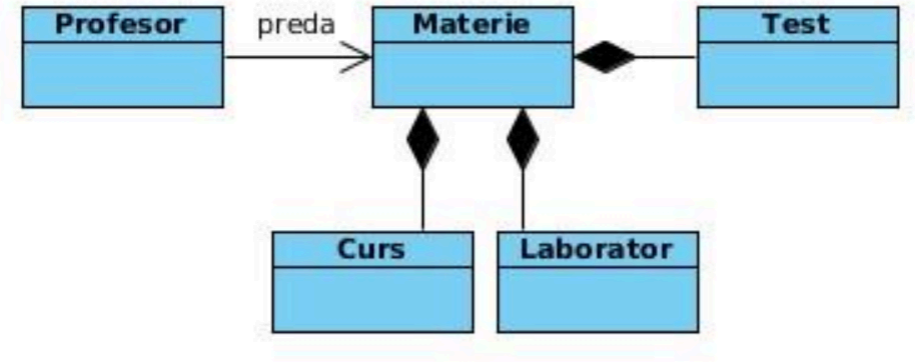
Selectați afirmațiile adevărate.
❌(a) Clasa Student moștenește clasa StudentBursier
[Clasa StudentBursier moștenește clasa Student și nu invers.]
✅(b) Un obiect de tip Student conține o colecție de obiecte de tip Proiect
[Între clasa Student și clasa Proiect este relație de compoziție.]
❌(c) Clasa Proiect are un atribut public de tip String
[Atributul de tip String este private.]
✅(d) Clasa Student are operația publică adaugăProiect
❌(e) Clasa Student are operația privată adaugăProiect
[Operația adaugăProiect este publică.]
✅(f) Clasa Student este superclasă pentru clasa StudentBursier

E5. Fie următoarea diagramă de clase.



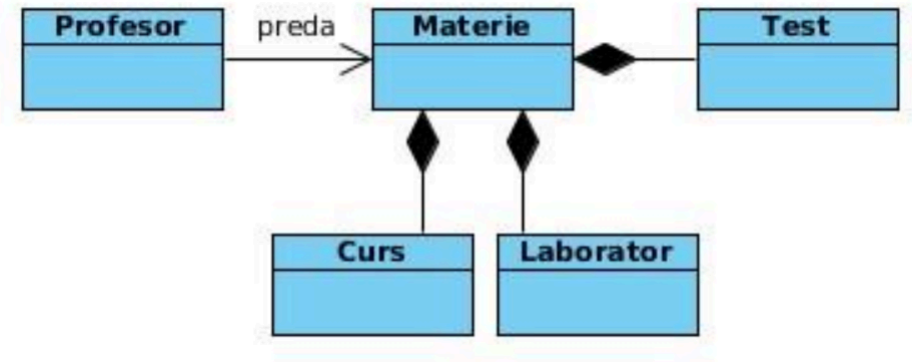
Care secvență de cod Java descrie corect relația dintre clasele Profesor și Materie?
❌(a) class Profesor extends Materie{...}
[Clasa Profesor nu este subclasă a clasei Materie. Cele două clase sunt în relație de asociere.]
✅(b) class Profesor {
private Materie preda; ...}
❌(c) class Materie {
private Profesor preda; ...}
[Relația de asociere este unidirecțională, adică „profesorul predă materia”, prin urmare, doar clasa Profesor poate avea ca atribut un obiect de tip Materie.]
❌(d) class Materie {
private Vector<Materie> preda;...}
[Doar clasa Profesor poate avea ca atribut un obiect de tip Materie, și nu invers. Relația dintre Profesor și Materie este de asociere, nu de compoziție.]

E6. Fie următoarea diagramă de clase.



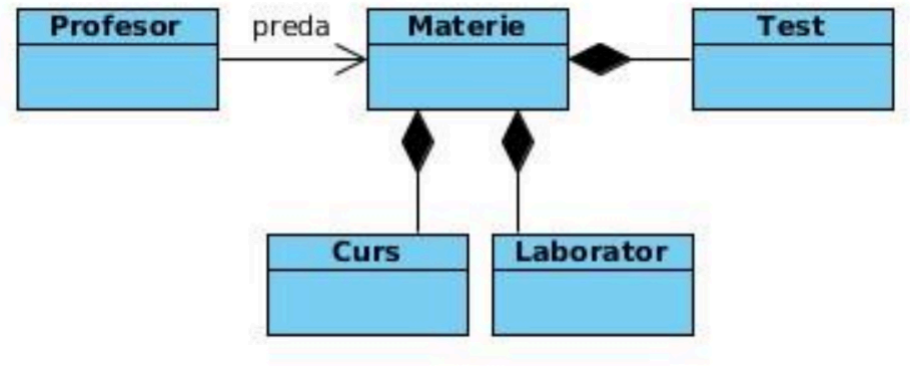
Care afirmații sunt adevărate?
✅(a) Clasa Materie definește o compoziție de obiecte de tip Curs
❌(b) Între clasa Profesor și clasa Materie există o asociere bidirecțională.
[Asocierea dintre Profesor și Materie este unidirecțională: „profesorul predă materia”.]
❌(c) Clasa Test moștenește clasa Materie.
[Relația dintre Materie și Test este de compoziție. Clasa Test nu este subclasă a clasei Materie.]
✅(❌)(d) Clasa Materie definește un agregat de obiecte de tip Laborator.
[Relația dintre Materie și Laborator este de compoziție, nu de agregare.]
✅(e) Un obiect de tip Materie conține o colecție de obiecte de tip Test

E7. Fie următoarea diagramă de clase.



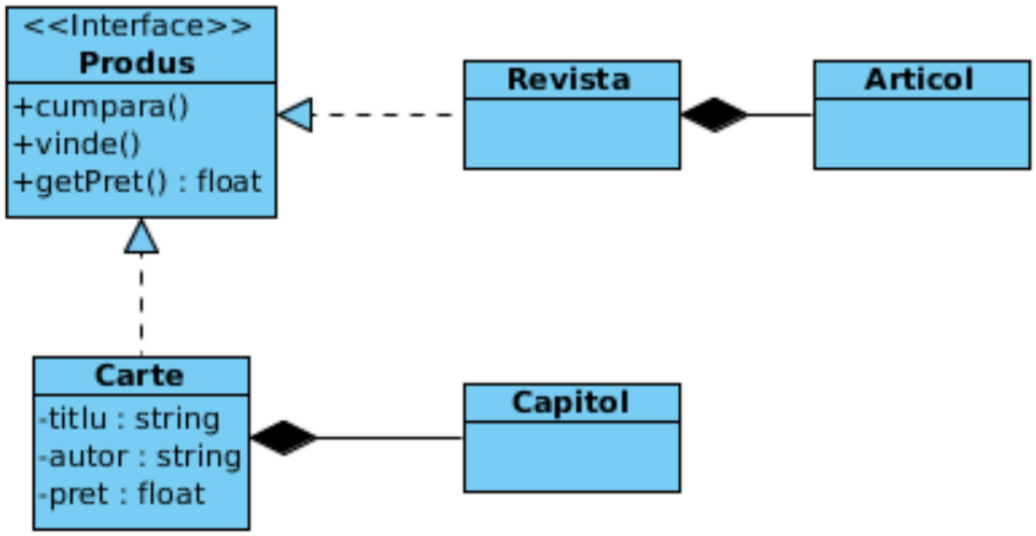
Care secvență de cod Java descrie corect și complet relația clasei Materie cu clasa Laborator?
❌(a) class Materie extends Laborator{...}
[Relația dintre Materie și Laborator nu este de moștenire, ci de agregare.]
❌(b) class Laborator extends Materie{...}
✅(c) class Materie {
private Collection <Laborator> laboratoare = new Collection();...}
class Laborator {
private Materie materie;
...}
[Clasa Materie are o colecție de obiecte de tip Laborator, iar clasa Laborator poate avea un atribut de tip Materie.]
❌(d) class Materie {
private Laborator laborator;...}
class Laborator {
private Collection<Materie> materie;
...}
[Clasa Materie conține o colecție de obiecte de tip Laborator și nu invers.]
❌(e) class Materie {
private Collection<Laborator> laboratoare;...}
class Laborator {
private Materie materie;
...}
[Colecția de obiecte de tip Laborator nu este instanțiată.]

E8. Fie următoarea diagramă de clase.



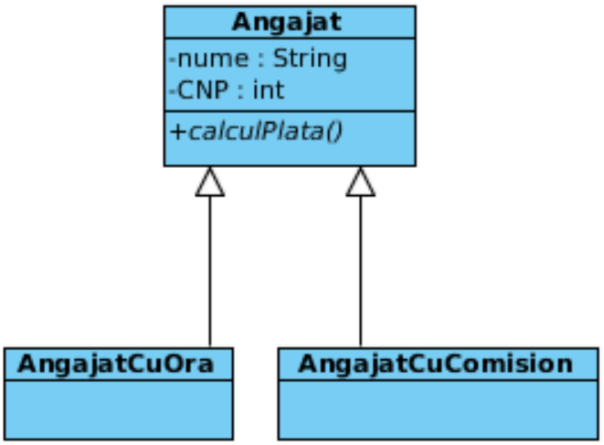
Selectați descrierea corectă și completă a relațiilor reprezentate în diagramă:
❌(a) Asociere între clasele Profesor și Materie; agregare între clasele Materie(agregat) și Test(componenta), Materie(agregat) și Laborator(componenta), Materie(agregat) și Curs(componenta).
[Relațiile dintre Materie și Curs/Test/Laborator sunt de compoziție, nu de agregare.]
❌(b) Asociere unidirecțională, numită preda, de la clasa Profesor la clasa Materie; agregare între clasele Materie(agregat) și Test(componenta), Materie(agregat) și Laborator(componenta), Materie(agregat) și Curs(componenta).
[Relațiile dintre Materie și Curs/Test/Laborator sunt de compoziție, nu de agregare.]
❌(c) Asociere unidirecțională, numită preda, de la clasa Profesor la clasa Materie; clasa Materie este superclasă pentru clasele Test, Laborator și Curs.
[Relațiile dintre Materie și Curs/Test/Laborator sunt de compoziție, nu de moștenire.]
✅(d) Asociere unidirecțională, numită preda, de la clasa Profesor la clasa Materie; compoziție între clasele Materie(compozit) și Curs(componenta); compoziție între clasele Materie(compozit) și Laborator(componenta); compoziție între clasele Materie(compozit) și Test(componenta).
❌(e) Asociere unidirecțională, numită preda, de la clasa Profesor la clasa Materie; clasele Curs, Laborator și Test moștenesc clasa Materie.
[Relațiile dintre Materie și Curs/Test/Laborator sunt de compoziție, nu de moștenire.]

E9. Fie următoarea diagramă de clase.



Care secvențe de cod Java sunt valide pentru clasa Carte?
❌(a) class Carte extends Produs{...}
[Produs nu este o clasă, ci o interfață. Nu e extends, ci implements.]
✅(b) private float pret;
❌(c) private pret float;
[Sintaxa nu este corectă.]
❌(d) public float getPret();
[Metoda nu este implementată, ci doar declarată. Prin realizarea interfeței Produs, clasa Carte trebuie să implementeze metodele interfeței realizate.]
✅(e) class Carte implements Produs{...}
❌(f) private float pret(){...}
[Nu există metodă pret(), ci există atributul pret.]
✅(g) public float getPret(){...}
❌(h) public Produs cumpara(){...}
[Metoda cumpara nu are return type Produs, nu este specificat.]
✅(i) private Collection<Capitol> capitole = new Collection();
[Între clasele Carte și Capitol este relație de compoziție. Clasa Carte are ca atribut o colecție/listă/tablou de obiecte de tip Capitol.]

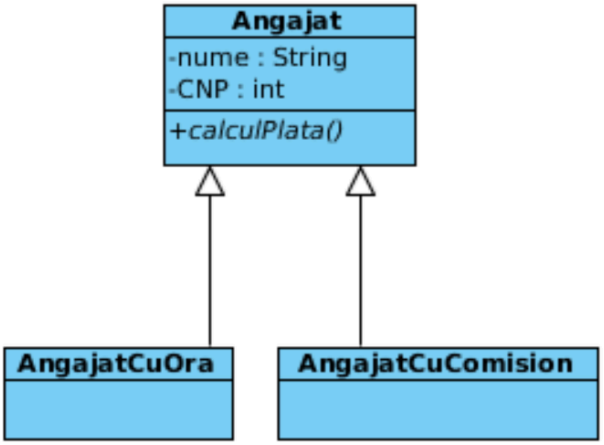
E10. Fie următoarea diagramă de clase.



Care secvență de cod Java definește corect și complet ceea ce rezultă din diagramă pentru clasa Angajat?
❌(a) class Angajat {
private String nume;
private int CNP;
public abstract void calculPlata();
...}
[Clasa ne-abstractă Angajat nu poate avea metoda abstractă calculPlata().]
✅(b) abstract class Angajat {
private String nume;
private int CNP;
public abstract void calculPlata();

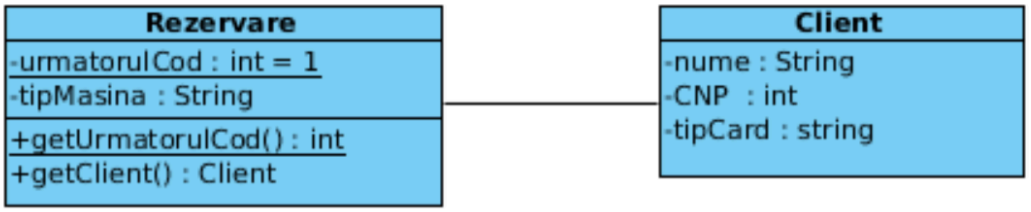
...}
❌(c) abstract class Angajat {
private String nume;
private int CNP;
public abstract void calculPlata(){...}
...}
[Metoda calculPlata() trebuie doar declarată, nu și implementată, atâta timp cât clasa Angajat este abstractă.]
❌(d) abstract class Angajat {
private String nume;
private int CNP;
public void calculPlata();
...}
[Clasa abstractă Angajat nu poate avea metoda ne-abstractă calculPlata().]

E11. Fie următoarea diagramă de clase.



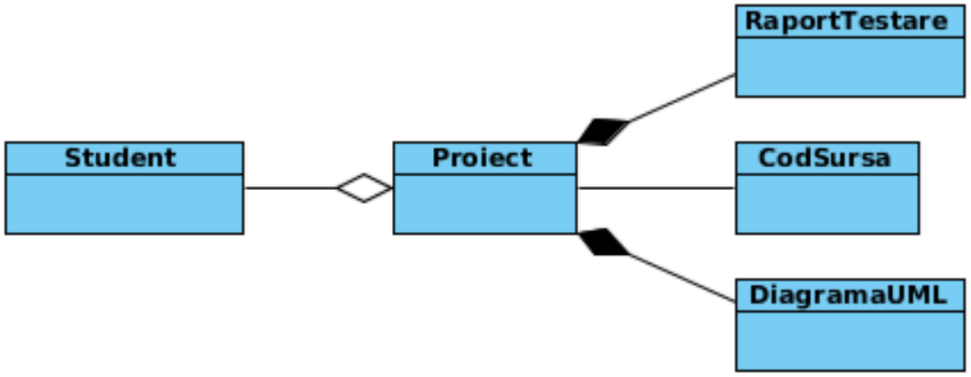
Care secvențe de cod Java sunt valide pentru clasa AngajatCuOra?
✅(a) class AngajatCuOra extends Angajat{...}
[Între clasele Angajat și AngajatCuOra este relație de moștenire: clasa AngajatCuOra moștenește clasa Angajat.]
❌(b) class AngajatCuOra implements Angajat{...}
[Angajat nu e interfață, ci clasă.]
❌(c) public calculPlata();
[Metoda calculPlata() este doar declarată, nu și implementată.]
✅(d) public calculPlata(){...};
[Metoda calculPlata este implementată.]

E12. Care secvențe de cod Java sunt valide pentru clasa Rezervare?



❌(a) class Rezervare extends Client{...}
[Clasele Rezervare și Client sunt în relație de asociere, nu de moștenire.]
❌(b) private int urmatorulCod = 1;
[Atributul urmatorulCod este static.]
✅(c) private static int urmatorulCod = 1;
✅(d) public static int getUrmatorulCod(){...};
✅(e) private Client client;
❌(f) public static Client getClient(){...};
[Sintaxa incorectă.]

E13. Fie următoarea diagramă de clase.



Selectați afirmațiile valide.
❌(a) Clasa Student definește un agregat de obiecte de tip Proiect iar clasele RaportTestare și DiagrameUML definesc compoziții de obiecte de tip Proiect.
[Clasa Proiect definește un agregat de obiecte de tip Student, și nu invers. Clasa Proiect definește compoziții de obiecte de tip RaportTestare și DiagrameUML, și nu invers.]
❌(b) Clasa Proiect definește compoziții de obiecte de tip Student, de tip DiagrameUML și de tip RaportTestare.
[Clasa Proiect definește compoziții doar de obiecte de tip DiagrameUML și de tip RaportTestare.]
✅(c) Clasa Proiect definește un agregat de obiecte de tip Student și compoziții de obiecte de tip DiagrameUML și de tip RaportTestare.
❌(d) Clasa Proiect definește o compoziție de obiecte de tip Student și agregate de obiecte de tip DiagrameUML și de tip RaportTestare.
[Clasa Proiect definește un agregat de obiecte de tip Student, și definește compoziții de obiecte de

tip RaportTestare și DiagrameUML.]
✅(e) Clasa Proiect este în relație de asociere cu clasa CodSursa.