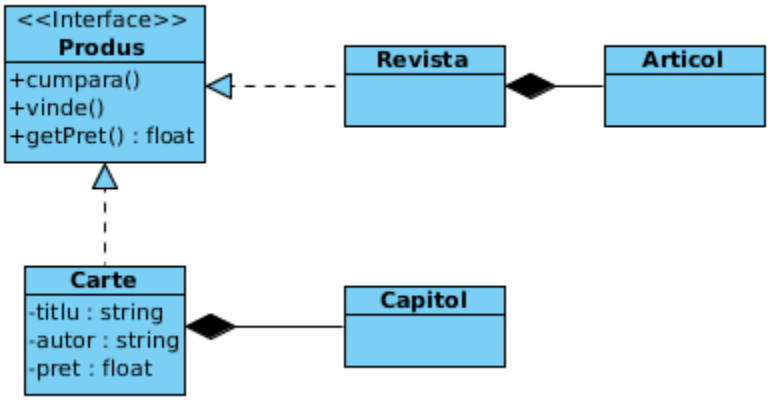
Pentru fiecare exercițiu indicați varianta(ele) corectă(e) și **explicați de ce fiecare din celelalte variante este incorectă**.

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`. Produs este interfata, nu clasa

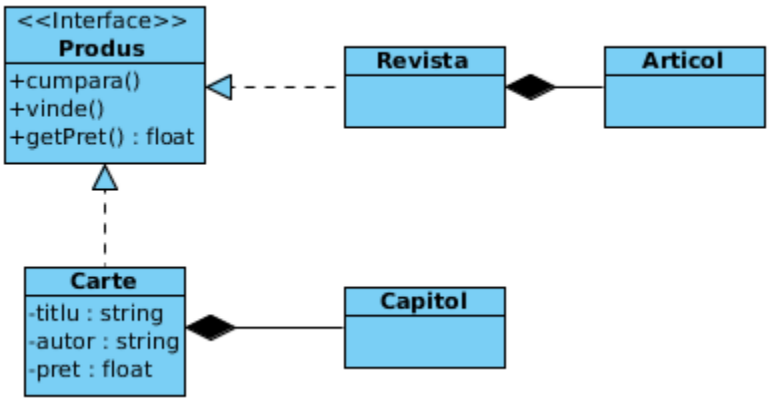
(b) Generalizare între clasa `Carte`(subclasă) și interfața `Produs`(superclasă); clasa `Carte` definește o compoziție de obiecte de tip `Capitol`. Nu este generalizare, si realizare

(c) Realizare între clasa `Carte` și clasa `Produs`; clasa `Carte` implementează clasa `Produs`; clasa `Carte` definește o compoziție de obiecte de tip `Capitol`. Produs nu este o clasa, ci o interfata

(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 defineste o comp de obiecte de tip Capitol

E2. Fie următoarea diagramă de clase.



Ce secvențe valide de cod Java rezultă din diagramă pentru clasa `Revista`?

(a) `class Revista extends Produs{...}` Trebuia implements

(b) `class Produs implements Revista{...}` Trebuia invers, Revista implements produs

(c) `private Collection<Articol> capitole = new Collection();`

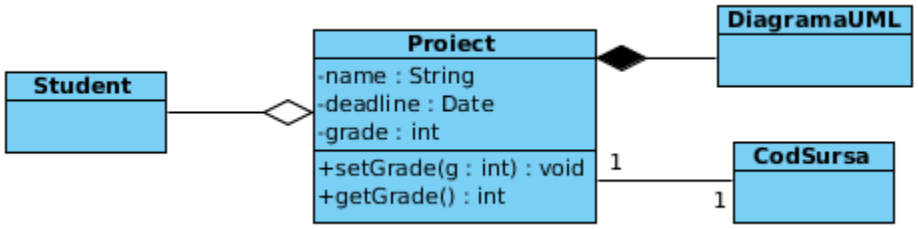
(d) `public float getPret();` Trebuie implementata

(e) `protected float getPret();` Trebuie public si nu e implementata

(f) `class Revista implements Produs{...}`

(g) `public Produs vinde(){...}` Tipul metodei vinte este de tip void, nu produs

E3. Fie următoarea diagramă de clase.



Care secvențe de cod Java sunt valide pentru clasa `Proiect`?

(a) `private CodSursa theCode;`

(b) `private grade int;` Sintaxa gresita. trebuia private int grade

(c) `private Collection<DiagramaUML> diagrams = new Collection();`

(d) `public Date deadline;` Trebuia private

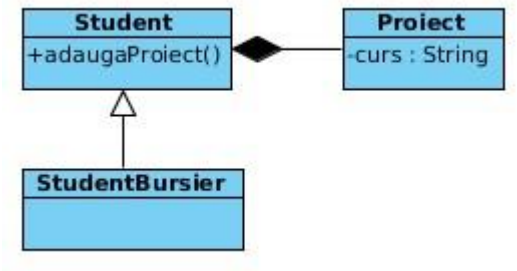
(e) `public getGrade(){...}` Trebuia tipul int

(f) `class Proiect extends Student{...}` este Agregare si nu Generalizare

(g) `public void setGrade(int g){...}`

(h) `private DiagramaUML diagram;` Trebuie mai multe obj de tip diagramaUml (trebuia un collection)

E4. Fie următoarea diagramă de clase.



Selectați afirmațiile adevărate.

(a) Clasa `Student` moștenește clasa `StudentBursier` Este total invers

(b) Un obiect de tip `Student` conține o colecție de obiecte de tip `Proiect`

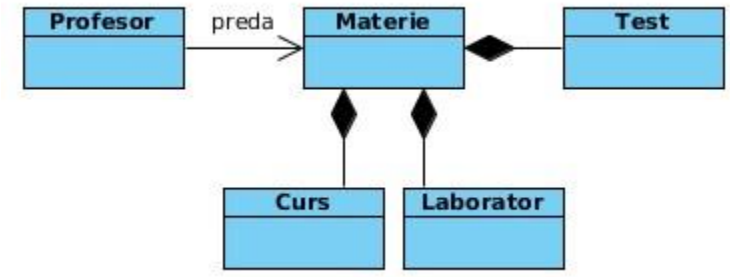
(c) Clasa `Proiect` are un atribut public de tip `String` Atributul e private

(d) Clasa `Student` are operația publică `adaugăProiect`

(e) Clasa `Student` are operația privată `adaugăProiect` Este operatie publica

(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`?

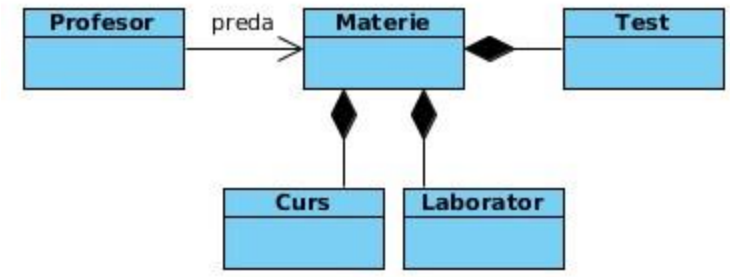
(a) `class Profesor extends Materie{...}` Este asociere unidim, nu generalizare

(b) `class Profesor {`
`private Materie preda; ...}` Sensul de asociere este gresit

(c) `class Materie {`
`private Profesor preda; ...}`

(d) `class Materie {`
`private Vector<Materie> preda;...}` Este asociere, deci nu este o lista de obiecte de tip Materie

E6. Fie următoarea diagramă de clase.



Care afirmații sunt adevărate?

(a) Clasa `Materie` definește o compoziție de obiecte de tip `Curs`

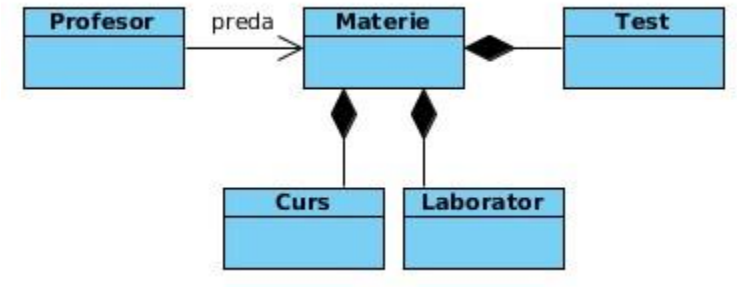
(b) Între clasa `Profesor` și clasa `Materie` există o asociere bidirecțională. E asoc. unidirectionala

(c) Clasa `Test` moștenește clasa `Materie`. Nu, este o compozitie

(d) Clasa `Materie` definește un agregat de obiecte de tip `Laborator`. E compozitie

(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{...}` Nu, este compozitie

(b) `class Laborator extends Materie{...}` E compozitie

(c)
```
class Materie {
  private Collection <Laborator> laboratoare = new Collection();...}

class Laborator {
  private Materie materie;
   ...}
```

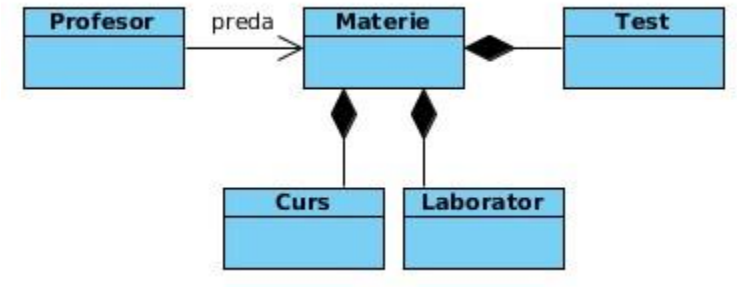
(d)
```
class Materie {
  private Laborator laborator;...}
```
Trebuie sa fie mai multe obiecte de tip Lab

```
class Laborator {
  private Collection<Materie> materie;
   ...}
```

(e)
```
class Materie {
  private Collection<Laborator> laboratoare;...}
```
Crearea Collection-ului nu e corect scrisa sintactic

```
class Laborator {
  private Materie materie;
   ...}
```

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). E compozitie intre Materie si Test

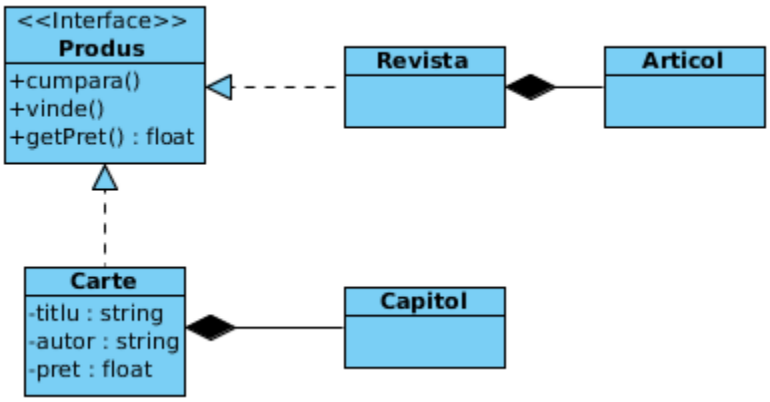
(b) Asociere unidirecțională, numită `preda`, de la clasa `Profesor` la clasa `Materie`; agregare între clasele `Materie`(agregat) și `Test`(componenta), `Materie`(agregat) și `Laborator`(componenta), `Materie`(agregat) și `Curs`(componenta). E compozitie

(c) Asociere unidirecțională, numită `preda`, de la clasa `Profesor` la clasa `Materie`; clasa `Materie` este superclasă pentru clasele `Test`, `Laborator` și `Curs`. Nu e superclasa pt Lab si Curs

(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`. Nu mostenesc

E9. Fie următoarea diagramă de clase.



Care secvențe de cod Java sunt valide pentru clasa `Carte`?

(a) `class Carte extends Produs{...}` Trebuia Implements

(b) `private float pret;`

(c) `private pret float;` Gresit sintactic

(d) `public float getPret();` Nu e implementata

(e) `class Carte implements Produs{...}`

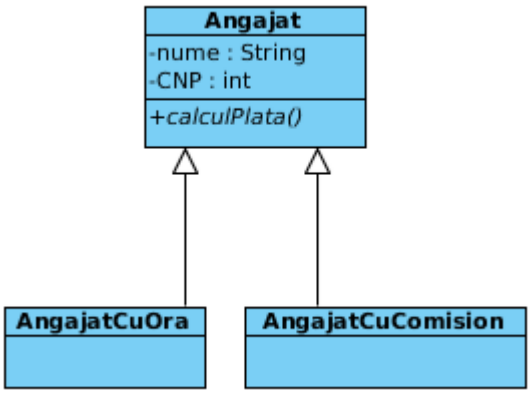
(f) `private float pret(){...}` Nu trebuie {...}

(g) `public float getPret(){...}`

(h) `public Produs cumpara(){...}` Trebuia tipul void si nu produs

(i) `private Collection<Capitol> capitole = new Collection();`

E10. Fie următoarea diagramă de clase.



Care secvență de cod Java definește corect și complet ceea ce rezultă din diagramă pentru clasa `Angajat`?

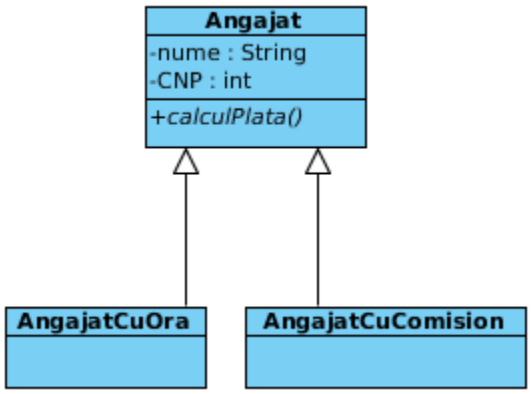
a)
```
class Angajat {
    private String nume;
    private int CNP;
    public abstract void calculPlata();
     ...}
```
Clasa trebuie sa fie si ea abstracta daca contine metoda abstr.

(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(){...}
     ...}
```
typo?

(d)
```
abstract class Angajat {
    private String nume;
    private int CNP;
    public void calculPlata();
     ...}
```
Trebuia sa fie metoda abstracta

E11. Fie următoarea diagramă de clase.



Care secvențe de cod Java sunt valide pentru clasa `AngajatCuOra`?

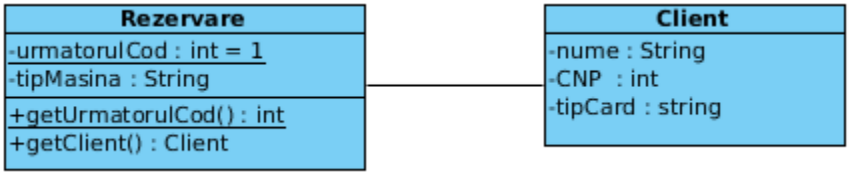
(a) `class AngajatCuOra extends Angajat{...}`

(b) `class AngajatCuOra implements Angajat{...}` Trebuia extends, e mostenire/Generalizare

(c) `public calculPlata();` trebuia void

(d) `public calculPlata(){...};`

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



(a) `class Rezervare extends Client{...}` E asociere, nu generalizare

(b) `private int urmatorulCod = 1;` Trebuia sa fie static

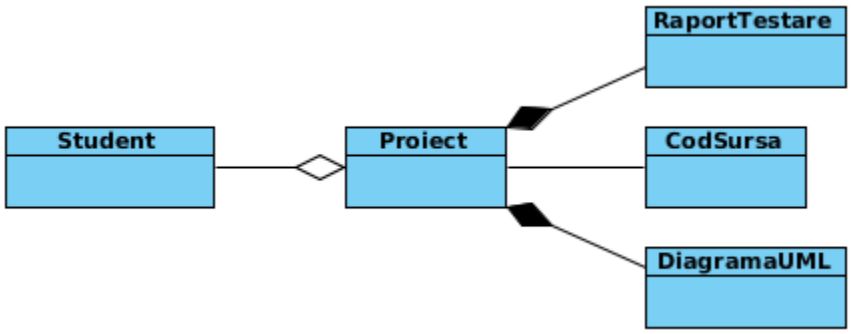
(c) `private static int urmatorulCod = 1;`

(d) `public static int getUrmatorulCod(){...};`

(e) `private Client client;`

(f) `public static Client getClient(){...};` Nu e statica

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`. Ordinea e inversa la toate

(b) Clasa `Proiect` definește compoziții de obiecte de tip Student, de tip `DiagrameUML` și de tip `RaportTestare`. Defineste agregare de obj de tip student

(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`. Agregat de obiecte de tip stud si compozitie de restul

(e) Clasa `Proiect` este în relație de asociere cu clasa `CodSursa`.

RASPUNSURI:

E1: d
E2: c, f
E3: a, c, g
E4: b, d, f
E5: b
E6: a, d, e
E7: c
E8: d
E9: b, e, g, i
E10: b
E11: a, d
E12: c, d, e
E13: c, e