Lecţia 13

Exerciţii şi probleme propuse spre rezolvare

1. Scrieţi o clasă *CC* care furnizează o metodă pentru aflarea numărului de instanţe ale clasei *CC* care s-au creat.

2. Avem clasa *Exemplu*. În ce condiţii putem scrie

   ```
   AltExemplu ae = new Exemplu(); ?
   ```

   Daţi două variante de răspuns şi explicaţi diferenţa/diferenţele dintre ele.

3. Codul următor nu poate fi compilat.

   ```
   interface A {
       void m1();            // 1
       public void m2();     // 2
       protected void m3();  // 3
       private void m4();    // 4
   }
   ```

   Specificaţi linia/liniile care cauzează necompilarea codului şi explicaţi pe scurt de ce se întâmplă acest fapt.

4. Un obiect poate avea mai multe tipuri?

5. Scrieţi o clasă *GrupaPOO* ce are un constructor public cu un parametru de tip *int* ce reprezintă numărul studenţilor din grupă. Implementaţi clasa astfel încât aceasta să poată fi instanţiată doar dacă numărul studenţilor din grupă este cuprins în intervalul [12-15].

6. Presupunem că în clasa *C* avem metoda

```
public void oMetoda throws Exceptie2, Exceptie3 {}
```

şi apelul ei

```
public static void main(String[] args) {
    try {
        C c = new C();
        c.oMetoda();
    }
    catch(Exceptie1 e1) {
        System.out.println("Exceptie1");
    }
}
```

Care e relaţia între clasele *Exceptie1*, *Exceptie2* şi *Exceptie3* astfel încât codul de mai sus să poată fi compilat?

7. Clasele *E1* şi *E2* sunt excepţii. Ce trebuie să fie îndeplinit pentru ca următorul cod să fie corect (Atenţie: nu se va modifica în niciun fel codul de mai jos !!!).

```
class SomeClass {
    public void foo(boolean b1, boolean b2) {
        if (b1 && b2)
            throw new E1();
        else
            throw new E2();
    }
}
```

8. Specificaţi relaţia/relaţiile de moştenire care există între clasele *Object*, *RuntimeException*, *Error*, *Throwable* şi *Exception*.

9. Se dă secvenţa de cod de mai jos:

```
class A {
    public static void main (String[] args) {
        Error error = new Error();
        Exception exception = new Exception();
        System.out.print((exception instanceof Throwable) + ",");
        System.out.print(error instanceof Throwable);
    }
}
```

- Care este rezultatul rulării acesteia?
- Daţi explicaţiile pentru care se produce acest rezultat.

10. Presupunem că într-o excepţie există o metodă a cărei nume va trebui setat de voi. Să se implementeze această excepţie astfel încât să existe o altă metodă care ne poate spune de câte ori a fost apelată metoda a cărei nume a fost setat de voi.

11. Scrieţi un exemplu în care să creaţi o excepţie verificată şi una neverificată. Explicaţi diferenţele dintre cele două tipuri de excepţii.

12. În clasa *C* avem metoda

```
public void oMetoda throws Exceptie1, Exceptie2, Exceptie3 {}
```

şi apelul ei

```
public static void main(String[] args) {
    try {
        C c = new C();
        c.oMetoda();
    }
    catch(Exceptie1 e1) {
        System.out.println("Exceptie1");
    }
    catch(Exceptie2 e2) {
        System.out.println("Exceptie2");
    }
    catch(Exceptie3 e3) {
        System.out.println("Exceptie3");
    }
}
```

Care e relaţia dintre clasele *Exceptie1*, *Exceptie2* şi *Exceptie3* dacă afirmaţia următoare e adevărată: indiferent de excepţia aruncată de metoda *oMetoda* pe ecran, întotdeauna se afişează "Exceptie1"?

13. Presupunem că avem pachetul *unu* în care sunt definite clasele publice *U1*, *U2* şi pachetul *unu.pp* în care sunt definite clasele publice *P1*, *P2*. În fişierul *F* din pachetul *poo* avem instrucţiunea *import unu.**. În aceste condiţii, în fişierul *F* se poate instanţia clasa *P1*?

14. De câte feluri pot fi specificatorii de acces ai unei clase în interiorul unui pachet? Daţi exemplu de o clasă care poate fi folosită doar în interiorul pachetului în care a fost declarată.

15. Presupunem că avem pachetul *unu* în care sunt definite clasele publice *U1* şi *U2* şi pachetul *unu.pp* în care sunt definite clasele publice *P1* şi *P2*, acestea neavând nici un constructor definit de programator. În fişierul *F* din pachetul *doi* avem instrucţiunea *import unu.**. În fişierul *F* avem o clasă *C*. Se poate scrie în interiorul unei metode a clasei *C* codul *P1 p = new P1()*?

16. Fie clasele de mai jos. Cum pot fi folosite instanţe noi ale clasei *PackageAClass* în metoda *foo()* din clasa *PackageBClass* fără a modifica clasa *PackageAClass* sau pachetul în care se află; pot fi adăugate clase noi.

```
//fisierul PackageAClass.java
package a;
public class PackageAClass {
    PackageAClass() { }
}

//fisierul PackageBClass.java
package b;
public class PackageBClass {
    PackageBClass() { }
    void foo() { }
}
```

17. Ce fel de clase pot fi folosite în afara pachetului în care au fost declarate? Ce fel de metode aparţinând unei clase dintr-un pachet nu pot fi folosite în afara pachetului în care a fost declarată clasa?

18. Avem o clasă *Elemente* care încapsulează o colecţie *ArrayList*. În această colecţie dorim să adăugăm obiecte periferice de tipul *Tastatura*, *Ecran* şi *Difuzor*. În acest scop am folosit următoarele metode pentru adăugare:

```
public void add(Tastatura t)
public void add(Ecran e)
public void add(Difuzor d)
```

    În viitorul apropiat s-ar putea să dorim să adăugăm şi alte tipuri de periferice cum ar fi un *Scanner* şi atunci ar trebui să adăugăm în clasa *Elemente* metoda

```
public void add(Scanner t).
```

    Propuneţi o soluţie pentru adăugarea de elemente în colecţie astfel încât adăugarea unui element de tip *Scanner* să nu implice modificări asupra clasei *Elemente*.

19. Avem o clasă *Persoana* care are două atribute: *nume(String)* şi *varsta(int)*. Dorim să reţinem obiecte de acest fel într-o colecţie de tip *SortedSet* ordonată crescător

după nume. În acest context implementaţi clasa *Persoana*, creaţi mai multe obiecte de acest tip, adăugaţi-le în colecţie şi precizaţi rezultatul afişării colecţiei.

20. Precizaţi utilitatea iteratorilor. Creaţi o colecţie în care adăugaţi elemente de tip *Persoana* (o persoană are un nume, o funcţie, un venit şi o vârstă). Folosind un iterator măriţi cu 1000 venitul persoanelor care au funcţia "programator".

21. În care din implementările următoare ale interfeţei *List* accesul la un element din mijlocul listei prin intermediul unui index se face mai încet şi de ce: *ArrayList* sau *LinkedList*?

22. Avem o clasă *Persoana* care are două atribute: *nume(String)* şi *cod numeric personal(String)*. Fiecare persoană are un cod numeric personal unic. Să se implementeze o metodă care returnează intersecţia a două colecţii de tip *Set* care conţin doar obiecte de tip *Persoana*.

23. Explicaţi câteva diferenţe dintre *Thread* şi *Runnable*.

24. Să se implementeze o clasă *ThreadManager* în care se pot înregistra thread-uri după nume şi se pot porni thread-uri folosind numele registrat:

```
ThreadManager m = new ThreadManager();
//t1,t2 - referinte spre doua threaduri
m.register(t1, "ThreadOne");
m.register(t2, "ThreadTwo");
m.startThread("ThreadTwo");
```

25. Modelaţi un fir de execuţie care numără de la 1 la 99, afişând pe ecran fiecare număr. Instanţiaţi şi porniţi firul.

26. Daţi un scurt exemplu în care să ilustraţi conceptul de producător /consumator.

27. Avem o *Memorie* ce poate stoca cel mult un şir de caractere (*String*). Există mai multe procesoare (fire de execuţie) care pot scrie în această memorie doar dacă elementul scris anterior de către un procesor a fost preluat de către un scriitor (fir de execuţie) ce citeşte la infinit din memorie, eliberând-o şi scriind conţinutul ei pe un support fizic. Procesorul poate scrie în memorie doar dacă aceasta este goală, în caz contrar el trebuie să aştepte eliberarea ei, iar dacă memoria este plină şi conţinutul ei este egal cu cel dorit a fi scris de processor, acesta abandonează scrierea în memorie. Să se implementeze clasa *Memorie*, comentându-se fiecare linie a sa.

28. Care e rezultatul execuţiei codului de mai jos? Detaliaţi răspunsul dat.

```
class ThreadOne extends Thread {

    public ThreadOne(Runnable r) {
        super(r);
    }

    public void run() {
        System.out.print("ThreadOne");
    }
}

class ThreadTwo implements Runnable {
    public void run() {
        System.out.print("ThreadTwo");
    }
}

class C {
    public static void main(String[] args) {
        new ThreadOne(new ThreadTwo()).start();
    }
}
```

29. Explicaţi de ce codul următor nu afişează pe ecran întotdeauna “LaboratorulPOO LaboratorulPOO....LaboratorulPOO”.

```
class T extends Thread {

    private void scrie() {
        System.out.print("Laboratorul");
        System.out.print("POO");
    }

    public void run() {
        int i = 100;
        while(i>0) {
            scrie();
            i--;
        }
    }
}
```

```
class C {
    public static void main (String[] args) {
        new T().start();
        new T().start();
    }
}
```

Modificaţi codul astfel încât acest lucru să se întâmple, dar să rămână cele două metode pentru afişarea pe ecran.

30. Se cere să se definească un ansamblu de clase necesare unei aplicaţii de evidenţă a unui depozit de produse. Un produs este caracterizat prin cod, nume, preţ de bază şi cantitate. Produsele sunt de două feluri: compensate şi necompensate. Preţul produselor compensate este obţinut prin înmulţirea preţului de bază cu un coeficient subunitar ce se transmite la crearea unui produs compensat iar al celor necompensate este chiar preţul de bază.

    Un client al depozitului trebuie să poată executa următoarele operaţii asupra depozitului:

    - adăugarea unui produs în felul următor: dacă produsul există în depozit, se actualizează doar cantitatea iar în caz contrar se măreşte numărul de produse existent cu o unitate (ATENTIE: produsele se vor reţine sub forma unui tablou iar în depozit se pot afla oricâte produse!).
    - tipărirea inventarului de produse la un moment dat (pentru fiecare produs existent în depozit se va tipări codul, numele, preţul şi cantitatea).

    Se cer:

    - diagrama UML pentru clasele necesare aplicaţiei.
    - implementarea claselor necesare.
    - o metodă main în care se va crea un depozit, se vor adăuga produse şi se va tipări inventarul depozitului creat.
    - ce s-ar întâmpla dacă în cadrul acestei aplicaţii nu s-ar folosi polimorfismul?
    - explicaţi mecanismul prin care la adăugarea unui produs se testează dacă produsul există sau nu în depozit.

31. Să se modeleze un sistem software simplificat orientat-pe-obiecte. Acesta este alcătuit din mai multe fişiere. Un fişier are un număr nelimitat de clase. O clasă are un anumit număr de linii de cod iar pe parcursul existenţei sale poate fi supusă unor modificări datorită unor probleme(bug-uri). Detalii despre toate bug-urile dintr-o clasă sunt stocate de către aceasta sub forma unei colecţii de Bug-uri, existând posibilitatea adăugării unui număr nelimitat de bug-uri unei clase. Fiecare fişier are asociat un autor. Atât clasa cât şi fişierul au asociate câte o denumire.

Bug-urile sunt de mai multe feluri:

- NullBug - acest tip de bug a apărut datorită folosirii unei referinţe ce nu a indicat spre nici un obiect. Afişarea acestui tip de bug pe ecran va determina mesajul *NullBug referinta*, unde referinţa este numele referinţei utilizate eronat. Numele referinţei va fi setat la instanţierea unui Bug de acest tip.
- CastBug - acest tip de bug apare datorită efectuării unei operaţii de conversie eronate dintre un tip şi alt tip. Acest tip de Bug se va tipări sub forma *CastBug referinta: Tipul ei - Tipul spre care s-a incercat conversia*. Toate valorile atributelor menţionate (trei) vor fi stocate sub formă de şiruri de caractere şi setate la crearea acestui tip de bug.

Pe lângă caracteristicile enunţate deja, entitatea Clasa mai are două servicii, unul furnizează numărul de linii de cod şi altul furnizează numărul total de bug-uri ce au apărut în cadrul clasei iar entitatea Fisier mai are o metodă ce furnizează numărul total de bug-uri existent în fişierul respectiv precum şi o altă metodă ce furnizează numărul de clase conţinute. Sistemul software are un număr nelimitat de fişiere. El, prin intermediul serviciilor, furnizează următoarele:

- numărul de fişiere conţinute
- numărul de clase conţinute
- numărul de autori distincţi ce au dezvoltat sistemul (un autor poate dezvolta mai multe fişiere)
- un serviciu pentru tipărire. Rezultatul acestui serviciu va informa clienţii clasei despre numărul de fişiere existente în sistem iar numele fiecărui autor distinct va fi furnizat împreună cu fişierele deţinute de acesta. Atenţie - numele unui autor apare o singură dată! Dacă un autor deţine mai multe fişiere, denumirile acestora se vor tipări pe câte o linie distinctă.

Entităţile Clasa, Fisier şi Sistem stochează sub formă de colecţii de obiecte numărul nelimitat de membri conţinuţi (Clasa are Bug-uri, Fisierul are Clase, Sistemul are Fisiere) şi fiecare entitate menţionată are un serviciu pentru adăugarea de elemente în propria colecţie.

Să se implementeze clasa sau, după caz, clasele ce modelează conform cerinţelor de mai sus bug-urile existente precum şi entităţile Clasa, Fisier şi Sistem. Să se instanţieze într-o metodă main un sistem cu câteva fişiere şi clase, unele clase având şi bug-uri ataşate.

Observaţie: Clasele, în afara serviciilor menţionate pe care acestea trebuie să le pună la dispoziţia utilizatorului, pot conţine oricâte atribute/servicii consideraţi că sunt necesare.

32. Să se scrie o clasă ce modelează conceptul de coadă, elementul ce este folosit pentru stocarea elementelor fiind tabloul. Elementele ce aparţin cozii la un moment dat sunt nişte carduri bancare, în cazul nostru Visa, Maestro şi MasterCard. Fiecare tip de card are imprimat numele utilizatorului (tip String, maxim 15 caractere), un număr format din 16 cifre(tip char[]) iar cardurile Visa şi MasterCard au un număr de control format din 3 cifre(tip char[]) inscripţionat pe spate. Toate valorile acestor atribute sunt setate prin intermediul constructorilor acestor clase. Dimensiunea cozii este setată prin intermediul unui constructor ce primeşte ca parametru un int. Dacă parametrul primit este negativ atunci acest constructor va genera o excepţie neverificată creată de voi. Clasa care modelează coada pune la dispoziţia unui client:
    - o metodă pentru introducerea unui element(card bancar) în coadă. Această metodă are ca unic parametru elementul ce se doreşte a fi introdus. Dacă nu mai există loc în coadă pentru introducerea unui nou element, clientul (obiectul care a apelat metoda de adăugare pusă la dispoziţie de coadă) va fi informat de faptul că momentan nu mai pot fi introduse elemente. Atenţie - această informare nu va fi realizată prin tipărirea unui mesaj iar tipul returnat de metodă va fi void!
    - o metodă pentru returnarea şi ştergerea următorului element(dintre elementele existente primul introdus) din coadă. Dacă coada este goală, acest lucru va fi semnalat clientului exact în modul în care a fost semnalat şi faptul că nu mai pot fi adăugate elemente.
    - o metodă pentru afişarea conţinutului cozii. Elementele vor fi afişate în ordinea PrimulIntrodus...UltimulIntrodus şi pentru fiecare card se vor tipări valorile tuturor atributelor conţinute însoţite de tipul cardului.
    - o metodă public boolean esteInCoada(Strategie strategie), unde Strategie e o interfaţă ce are metoda public boolean conditie(CardBancar card). Aceasta metodă (esteInCoada) returnează true dacă există cel puţin un element în coadă care satisface condiţia definită de strategie. Concret, parcurge toate elementele existente în coadă si dacă există cel puţin un element pentru care apelul metodei conditie a strategiei (evident, având ca parametru elementul curent (cardul curent)) returnează true, metoda esteInCoada returnează true. Dacă un astfel de element nu este identificat, metoda va returna false.

    Se cer:

    - Să se implementeze conform cerinţelor clasa Coada împreună cu toate clasele şi subclasele folosite de aceasta.
    - Să se implementeze interfaţa Strategie de către două clase, astfel încât să existe:
        - O clasă ce primeşte printr-un constructor un String corespunzător unui nume. În acest caz metoda public boolean conditie(CardBancar card) returnează true dacă numele utilizatorului primit ca parametru prin constructor este egal cu cel stocat în cadrul obiectului tip CardBancar referit de către parametru.

  - O clasă pentru care metoda public boolean conditie(CardBancar card) returnează true dacă numărul cardului conţine de trei ori cifra 5.
- Să se creeze într-o metodă main o coadă. Să se apeleze toate metodele puse la dispoziţie de către obiectul creat.
- Să se explice care sunt avantajele folosirii moştenirii de tip în cadrul acestei aplicaţii.
- Descrieţi modificările necesare astfel încât pe parcursul existenţei programului clasa Coada să nu poată fi instanţiată decât o singură dată. Apoi modificaţi implementarea clasei astfel încât aceasta să nu poată avea mai mult de o instanţă.

33. Fie o scenă grafică care conţine diferite tipuri de figuri. Concret, putem avea figuri de tip Cerc, Patrat şi Triunghi. Fiecare figură are o culoare, conţine o metodă pentru calculul perimetrului şi poate fi afişată sub următoarea formă: Tip Figura, Culoare, Perimetru (Culoare desemnează valoare culorii(ex: galben, roşu) iar Perimetru valoarea perimetrului). Se consideră că două figuri sunt egale dacă au acelaşi tip şi acelaşi perimetru. Figurile de tip Cerc au o rază, cele de tip Pătrat au o latură iar cele de tip Triunghi au trei laturi. Toate figurile au metode pentru modificarea atributelor prezentate anterior.

    Atunci când se instanţiază, fiecare figură se înregistrează la un observator. Un observator este o clasă ce are o singură instanţă şi conţine următoarele metode:

    - o metodă pentru adăugarea oricărui tip de figură (Cerc, Patrat sau Triunghi) într-o colecţie de obiecte, colecţia fiind un atribut al clasei.
    - o metodă prin care se anunţă observatorul că starea unei figuri conţinute (culoarea şi/sau raza/latura/laturile) s-a modificat. Apelul acestei metode e realizat de către toate tipurile de figuri atunci când se modifică caracteristica/caracteristicile acestora şi determină afişarea tuturor figurilor monitorizate pe ecran.
    - o metodă pentru afişarea tuturor figurilor ce sunt monitorizate.

    Se cer:

    - Să se implementeze conform cerinţelor ierarhia de figuri precum şi clasa ce modelează conceptul de observator.
    - Să se creeze într-o metodă main cel puţin o figură de fiecare tip, iar cel puţin uneia dintre ele să i se modifice un atribut. Se va exemplifica rezultatul rulării metodei main.
    - Să se explice care sunt avantajele folosirii polimorfismului în cadrul acestei aplicaţii.

34. Poliţia comunitară acordă diferite tipuri de amenzi pentru următoarele categorii de fapte, după cum urmează:

    - parcarea ilegală a maşinii – amendă fixă 500 RON

- aruncatul gunoaielor pe jos – amendă 200 RON * factorZonal, unde factorZonal este un coeficient cuprins între 1..4, corespunzător zonei în care s-a produs fapta
- distrugerea bunurilor din parcuri – amendă 300 RON * valoarea bunului

Fiecare tip de amendă are cel puţin următoarele caracteristici:

- conţine codul numeric personal (CNP) al cetăţeanului căruia a fost aplicată
- are o metodă ce returnează valoarea amenzii
- poate fi afişată sub următoarea formă: CNP Tip Amenda, Valoare. Tip Amenda desemnează cauza pentru care amenda a fost acordată iar Valoare reprezintă suma care trebuie încasată de la cetăţean.

În cadrul sistemului folosit există o clasă pentru gestiunea tuturor amenzilor acordate. Această clasă conţine:

- o colecţie de obiecte pentru stocarea tuturor amenzilor acordate
- o metodă pentru introducerea unei noi amenzi (de orice tip)
- o metoda public int valoare(Strategie strategie), unde Strategie e o interfaţă ce are metoda public boolean conditie(Amenda amenda). Această metodă (int valoare(Strategie strategie)) returnează suma amenzilor corespunzătoare elementelor din colecţie ce satisfac proprietatea definită de strategie. Concret, parcurge toate elementele colecţiei şi pentru fiecare element apelează metoda conditie a strategiei (evident, având ca parametru elementul curent (amenda curentă)) şi dacă aceasta returnează true, adaugă valoarea amenzii la suma ce va fi returnată.

Se cer:

- Să se implementeze conform cerinţelor ierarhia de amenzi precum şi clasa ce gestionează amenzile acordate.
- Să se implementeze interfaţa Strategie de către diferite clase, astfel încât să existe:
  - o clasă ce primeşte printr-un constructor un String corespuzător unui cod numeric personal. În acest caz metoda public boolean conditie(Amenda amenda) returnează true dacă codul numeric personal primit ca parametru prin constructor este egal cu cel stocat în cadrul obiectului tip Amenda referit de către parametru.
  - o clasă pentru care metoda public boolean conditie(Amenda amenda) returnează true dacă amenda a fost generată dotorită nerespectării semnului *Parcarea Interzisa*.
- Să se exemplifice printr-o metodă main funcţionarea sistemului. Se vor crea diferite tipuri de amenzi (cel puţin câte una de fiecare tip) şi se va returna suma amenzilor datorate de cetăţeanul ce are codul numeric personal *1031274456709*.

- Să se explice care sunt avantajele folosirii polimorfismului în cadrul acestei aplicaţii.

35. Într-un sistem simplificat de execuţie de sarcini (nu are legătură cu firele de execuţie) există mai multe feluri de task-uri. Indiferent de tipul său concret, pe un obiect task se poate apela din exterior:
   (a) o metodă denumită *execute* ce întoarce sub formă de întreg timpul de execuţie al task-ului în milisecunde (presupunem că metoda şi execută cumva fizic task-ul dar pentru simplitate pe noi ne interesează doar timpul de execuţie)
   (b) o metodă care întoarce reprezentarea sub formă de şir de caractere a obiectului task. Calculul timpului de execuţie şi reprezentarea sub formă de şir de caractere se realizează funcţie de tipul concret al task-ului.

   Tipurile concrete de task-uri sunt:

   - SimpleTask - un astfel de obiect primeşte la creare (prin constructor) un întreg reprezentând timpul de execuţie al task-ului. Pentru acest fel de task reprezentarea sub formă de şir de caractere este SimpleTask(timp_executie) unde timp_executie e întregul dat prin constructor. Valoarea acestui întreg poate fi modificată printr-o metodă changeTime (prin parametrul ei) definită în această clasă.
   - ConditionalTask - un astfel de obiect primeşte prin constructor două obiecte task ce pot fi de orice fel concret. Timpul de execuţie pentru un astfel de tip de task se calculează ca 5 milisecunde la care se adaugă aleator fie timpul de execuţie al primului task fie timpul de execuţie a celui de-al doilea (trebuie ţinut cont că timpul de execuţie al unui task condiţional poate diferi de la o execuţie la alta; pentru acest aleatorism puteţi utiliza metoda statică random din clasa Math). Pentru acest fel de task reprezentarea sub formă de şir de caractere este ConditionalTask(r1,r2) unde ri(i=1..2) este reprezentarea sub formă de şir de caractere a task-ului i dat prin constructor.
   - BlockTask - reprezintă o înşiruire de task-uri de orice fel (simple, condiţionale, blocuri) într-un număr nelimitat. Timpul de execuţie pentru un astfel de tip de task se calculează ca suma timpului de execuţie pentru fiecare task conţinut. Clasa mai pune la dispoziţie o metodă insert prin intermediul căreia se poate adăuga la un task bloc (din exteriorul unui astfel de obiect) un task de orice fel (prin intermediul parametrului metodei). Reprezentarea sub formă de şir de caractere are forma Block(r1,r2,..) unde ri(i=1,2,..) este reprezentarea sub formă de şir de caractere a task-ului i conţinut de task-ul block.

   Se cer:

   - Implementarea claselor descrise şi a altora dacă sunt necesare.
   - Să se construiască într-o metodă main (şi ţinând cont de implementarea claselor anterioare) un obiect task a cărui reprezentare sub formă de şir de caractere este
     ConditionalTask(SimpleTask(155), BlockTask(SimpleTask(5), SimpleTask(4))).

- Să se implementeze un mecanism prin care să putem afla în câte taskuri compuse (adică condiţionale sau bloc) a fost inclus un obiect task. În acest scop se va introduce corespunzător în clasa / clasele de task-uri o metodă care să întoarcă această valoare.

Notă: Schimbarea timpului de execuţie a unui task simplu, a unui task block (prin inserarea la ea de alte task-uri), etc., trebuie să fie observabilă în timpul de execuţie a altor task-uri compuse ce includ task-urile anterioare.

36. Într-un program de gestiune a angajaţilor unei firme există mai multe feluri de angajaţi. Fiecare obiect angajat este caracterizat de numele său şi are o metodă calculSalar care întoarce un double reprezentând salariul acelui angajat. Tipurile concrete de angajaţi şi modul specific de calculare a salariilor (pe lângă alte informaţii) sunt următoarele:

    - AngajatCuSalarFix - la crearea unui astfel de obiect se dă numele angajatului şi un double reprezentând salariul său fix. Aceste date vor fi memorate în starea obiectului. În cazul unui astfel de angajat, metoda calculSalar întoarce valoarea salariului dată la crearea obiectului. Clasa mai defineşte o metodă denumită schimbaSalarFix prin intermediul parametrului căreia se poate schimba valoarea salariului fix specificat la crearea obiectului.
    - AngajatCuOra - la crearea unui astfel de obiect se dă numele angajatului şi un double reprezentând salariul pe oră primit de angajat. Aceste date vor fi memorate intern în starea obiectului. Clasa mai defineşte metoda adaugaOre care primeşte ca parametru un double reprezentând un număr de ore lucrate. Toate orele lucrate vor fi memorate intern într-un obiect de acest fel, într-un tablou de maxim 31 de intrări (se presupune că nu se adaugă mai mult de 31 de ore). Tot această clasă defineşte metoda schimbaSalarPeOra prin intermediul parametrului căreia se poate schimba salariul pe oră setat iniţial (se poate da altă valoare prin intermediul parametrului metodei). În cazul unui astfel de angajat metoda calculSalar întoarece ca valoare numărul total de ore (suma orelor din tablou) înmulţită cu salariul pe oră.

    Sistemul mai defineşte o clasă Firma. Aceasta defineşte o metodă angajeaza prin care un angajat de orice fel (dat ca parametru metodei) este memorat ca angajat al acelei firme. Angajaţii unei firme sunt memoraţi intern într-un tablou de maxim 1024 intrări. Dacă se încearcă angajarea a mai mult de 1024 de angajaţi, metoda întoarce valoarea -1. Mai mult, metoda întoarce valoarea -2 în cazul în care se încearcă adăugarea aceluiaşi angajat la aceeaşi firmă (un angajat nu poate fi adăugat de două ori la aceeaşi firmă). Se consideră că două obiecte angajat sunt egale atunci când numele angajaţilor sunt egale (reprezintă aceleaşi secvenţe de caractere). În situaţia în care angajarea se realizează cu success metoda angajeaza întoarce valoarea 0.

Clasa Firma mai defineşte o metodă salariuMediu care întoarce un double reprezentând salariul mediu calculat pentru toţi angajaţii săi (media salariilor). Dacă obiectul de tip Firma apelat nu are nici un angajat, metoda întoarce valoarea 0.

Se cer:

- Implementarea claselor descrise mai sus şi a altora necesare.
- O metodă main în care se instanţiază câte două obiecte angajat de fiecare fel şi o firmă. Se adaugă obiectele angajaţi la firma creată şi se calculează (şi afişează) salariul mediu pentru acea firmă. Se va exemplifica apoi cazul în care se încearcă angajarea aceleiaşi persoane de mai multe ori la aceeaşi firmă. Se va schimba apoi salariul fix sau pe oră a unui angajat din cei creaţi anterior şi se va reafişa salariul mediu pentru acea firmă.

Notă: Se pot adăuga şi noi membri la clasele amintite mai sus dacă se consideră necesar. Schimbarea valorii salariului unui angajat trebuie să fie vizibilă dintr-un obiect Firma care conţine obiectul angajat menţionat anterior.