1. A)

class AltaPersoana

{

public void platestePrinAltaPersoana(String beneficiar, double suma)

{

/\*...\*/

}

}

/\*

In acest caz ar trebui sa se modifice metoda "plateste" din clasa Persoana adaugand inca un bloc if-else

care sa verifice cu instanceof daca obiectul continul de clasa persoana este o instanta a clasei "AltaPersoana"

Prin urmare, codul clasei Persoana este deschisa la modificari, prin urmare, se incalca principiul open-closed.

\*/

B)

class Persoana

{

ModalitatePlata o;

public Persoana(ModalitatePlata o)

{

this.o = o;

}

public void plateste(String contDest, double suma)

{

o.plateste(contDest, suma);

/\*

Acest program respecta principiul "open-closed", datorita acestui fapt

se pot adauga noi si noi modalitati de plata fara a trebui sa fie modificata

clasa Persoana in vreun fel sau altul

\*/

}

}

abstract class ModalitatePlata

{

public abstract void plateste(String beneficiar, double suma);

}

class Card extends ModalitatePlata

{

@Override

public void plateste(String beneficiar, double suma)

{

}

}

class OrdinPlata extends ModalitatePlata

{

@Override

public void plateste(String beneficiar, double suma)

{

}

}

1. A)

Pe linia marcata cu “1” se va tipari 56, deoarece:

Referinta a va primi o instante de forma: “Grup(12, 92)”,

Cand se apeleaza metoda “apelare” pe prma linie a metodei, obiectul spre care

Pointeaza referinta a este modificata in: “Grup(56, 0)”. Pe a doua linie se va modifica cel

De al doilea obiect transmis ca parametru, dar campul t al clasei Grup fiind statica, acesta se modifica si in obiectul referit de a. Deci, obiectul final referit de a va ave forma: “Grup(56, 33)”.

B) Este vorba despre cel de al doilea element al obiectului referit de a. Bazandu-ma pe explicatia data mai sus, afirm ca pe linia “2” se va tipari valoarea 33.

1. Un se vede
2. A) 1

B) se compileaza, frata

1. A) Se va tipari “A” pe ecran, deoarece metoda “ex” din clasa B un suprascrie metoda “ex” din clasa A, pentru ca:

B chiar este un subtip al clasei Object, dar chiar si in acest caz, suprascrierea de metoda un se poate face. In conditiile in care s-ar permite acest lucru, s-ar putea trimite orice fel de obiecte catre metoda ex si evident ex(B o) un suporta orice fel de obiecte. Astfel, s-ar crea o eroare la runtime.

B) Gresit, daca o clasa este abstracta, aceasta un s-ar putea instantia cu “new”, dar pot exista oricate referinte de tipul clasei abstracte. Deci, indiferent daca clasa A ar fi abstracta sau nu, programul ar rula in acelasi fel si un s-ar crea eroare de compilare.

1. A) Gresiiit. Intradevar instantierea cu new se poate face de forma: “new OriceSubclasaAInterfeteiList<A sau o subclasa a lui A>()”, dar cand vine vorba de adaugare, aceasta un se mai poate face, se va genera eroare de compilare. Forma aceasta data in exemplu se foloseste doar cand dorim sa accesam continutul unui obiect de tip List, nu si cand dorim sa modificam lista.

B) Gresit, obiectul referit de o poate sa fie o instante a clasa “OClasa” dar acesta poate sa fie si o clasa derivata din clasa “OClasa”. Daca se doreste ca obiectul sa fie strict o instanta a clasei OClasa, atunci se foloseste reflectia in forma urmatoare: return OClasa.class == o.getClass();