1.

a) Afirmatia este falsa. Intr-adevar, se va crea o instanta la executia “new A()”, dar in interiorul

obiectului nu se va crea a doua instante, deoarece aceasta contine doar o refertina catre un obiect A, nu si instantierea sa.

Mentionez ca este modul corect de a declara o instante in clasa, pentru ca daca un loc e linia “protected A” as fi avut “protected A = new A()”, la instantierea obiectului s-ar fi creat o eroare de tip StackOverflow, deoarece s-ar fi creat instante in continuu pana cand s-ar fi epuizat capacitatea stivei.

b) Da, este posibila instantierea folosind reflexia.

O instantiere a clasei ar arata in felul urmator: A a = (A)Class.forName(“A”).newInstaince();

Va trebui adaugat un bloc try-catch pentru ca instantierea de sus poate genera exceptii.

2.

a) Nu se va tipari 0 la sfarsitul executiei, deoarece exceptia aruncata de exec(int k) este aruncata atunci cand k = 3 si in acest moment se incheie executua blocului try si se trece in blocul catch corespunzator exceptiei care s-a aruncat. Prin urmare, un se va tipari 0, se va tipari 3.

b) Afirmatia este falsa. SpecialCase este o exceptie verificata, deci fara un block try-catch, nu poate fi folosita nicio metoda sau constructor care arunca acea exceptie. Bineinteles, se mai poate folosi clauza throws pentru a evita folosirea blocului try-catch, dar totusi, daca s-ar elimina blocul catch pur si simple din program, acesta un ar mai fi compilabil.

3. a)

Pe linia marcata cu “1” se va tipari 12 pentru ca aceata este prima valoare a perechii

care a fost setta de la inceput prin metoda set. In metoda apelare referita e va pointa catre

referinta f dar aceastea sunt referinte complet diferite de x si y (parametri transmisi metodei),

deci, obiectul referit de y va fi modificat, iar obiectul referit de y va ramane intact, la fel si referinta care a pointat initial catre ea.

b) Pe linia marcata cu “2” se va tipari 17 deoarece: la inceput, prin metoda set, se va instantia obiectul cu valorile 10 si 11. O alta referinta, z, va pointa catre acelasi obiect. Dupa, obiectul referit de y va fi modificat in (14, 15) de catre metoda “apelare”, iar acest obiect va mai fi odata modificat de catre referinta z prin metoda set in (16, 17). Deci, cel de al doilea element al perechii este 17.

4. a)

Intr-adevar, mostenirea un este solutia buna, deoarece o instante Coada va putea apela si metodele care sunt exclusiv pentru o lista. Deci, coada va putea sa extraga elemente de oriunde din lista continuta, nu numai de la inceput, cum s-a dorit.

Coada coada = new Coada();

coada.introdu( 1 );

coada.introdu( 2 );

coada.introdu( 3 );

coada.isSiEliminaPozitia( 1 ); /\* Executandu-se linia aceasta de cod, va incalca conceptul

tipului de data abstract, coada\*/

b) Modul potrivit de implementare nu este folosirea mostenirii, ci folosirea clasei “ListaMea” in noua clasa (“has a”).

}

public class Coada {

private ListaMea l;

public Coada()

{

l = new ListaMea();

}

public void introdu(Double i)

{

l.puneUltimul( i );

}

public Double extrage()

{

return l.iaSiPuneUltimul( 0 );

}

}

5. a)

Se poate apela si fara cast deoarece a fost folosita mostenirea si toate clasele publice si protejate, vor putea fi folosite de clasa C, prin urmare si metoda “public void sum(int a)”.

b) Faaals. La instantierea clase C in main se apeleaza si acel constructor pentru ca clasa B este superclasa clasei C. Nu este nevoie sa fie apelat acel constructor pentru ca este un constructor fara argumente, dar asta nu inseamna ca un se apeleaza deloc in constructorul clasei C. In cazul in care o clasa extinde alta clasa care are un contructor default, un mai este nevoie ca in constructorul clasei, sa se apeleze superconstructorul, deoarece apelul acestuia este introdus de compilator in caz ca acesta un apare.

6. a)

Faaals. La compilare nu se stie ce metoda va fi folosita in cazul legarii dinamice. Acest lucru se determina la runtime. Aceasta proprietate face posibilaextinderea facila a programelor object oriented, deoarece se pot adauga nenumarate clase deritate la o superclasa care sa contina o anumita metoda suprascrisa, fiecare avand propia implementare. Partea buna este ca metoda din superclasa un va trebui modificata niciodata, deoarece se pot crea cmetode care sa suprascrie acea metoda din clase derivate.

b) Afirmatia este falsa. Intr-o unitate de compilare poate sa existe o singura clasa cu extensia public, care poate fi folosita in exteriorul pachetului. Deci, daca clasele A si B sunt in aceeasi unitate de compilare, acestea un vor putea sa aiba amandoi specificatorul de acces “public”, prin urmare una dintre clase un va putea fi folosita in afara.

Deci, pentru ca clasele A si B sa poate fi folosite in afara pachetului, ele trebuie sa fie in unitati de compilare diferite.