1. A)

import java.util.\*;

public class Executor

{

public void transmiteMesaje(ArrayList<String> mesaje, ArrayList<String> destinatari, ArrayList modalitati)

{

int k = 0;

for(Object tmp : modalitati)

{

if(tmp instanceof Sms)

((Sms)tmp).sendSms(mesaje.get( k ), destinatari.get( k++ ) );

else if(tmp instanceof Fax)

((Fax)tmp).sendFax(mesaje.get( k ), destinatari.get( k++ ) );

else{

System.out.println("A treia lista contine elemente straine ");

System.exit( -1 );

}

}

}

}

B)

import java.util.\*;

interface Sender

{

public void sendMessage(String dest, String msg);

}

class Sms implements Sender

{

public void sendMessage(String dest, String msg)

{

}

}

class Fax implements Sender

{

public void sendMessage(String dest, String msg)

{

}

}

public class Executor

{

public void transmiteMesaje(ArrayList<String> mesaje, ArrayList<String> destinatari, ArrayList<Sender> modalitati)

{

int k = 0;

for(Sender tmp : modalitati)

{

tmp.sendMessage(mesaje.get( k ), mesaje.get( k++ ) );

}

}

}

1. A)

Linia marcata cu “1” va tipari pe ecram valoarea 20 pentru ca:

La inceput se vor crea doua obiecte: Pereche(70, Pereche(80, null)) si Pereche(90, Pereche(100, null)). Z va pointa catre primul obiect.

Dupa apelarea metodei “apelare”, linia “f,set(20, new Pereche(30 , null))” va modifica

Obiectul in: Pereche(20, Pereche(30, null)) deoarece referinta f va ajunge sa pointeze spre acelasi obiect spre care pointeaza si z. NU se mai modifica obiectul pana la incheierea programului, deci prima valoare a perechii va fi 20.

B)

Se va tipari pe ecran valoarea 80 la linia marcata cu “2” deoarece:

Obiectul initial spre care pointeaza y este: Pereche(90, Pereche(100, null)).

Dupa apelarea metodei “apelare” prima linie a metodei va modifica obiectul in:

Pereche(10, Pereche(80, null)), perechea din interiorul acestei perechi este luata din primul

Parametru transmit metodei.

Un se mai modifica obiectul pana la incheierea programului si valoarea tiparita pe ecran va fi 80.

1. A)

Pe linia marcata cu “1” se va tipari valoarea 34, deoarece:

Referinta x este de tip X care va primi o instante a subclasei Z.

Pe linia marcata cu “1” se apeleaza metoda “proc” de pe o referinta a clasei X. In obiectul Z nu am suprascriere a metodei “proc”, deci se va cauta aceasta metoda in subclasa Y, unde aceasta exista.

B)

Pe linia marcata cu “2” se va tipari pe ecran valoarea 14, deoarece:

La instantierea lui Z se vor autoapela constructorii superclaselor, pana se ajunge in clasa Object, dupa, valorea v din clasa x primeste valoarea 0, care va fi incrementata tot in aceasta clasa cu cu 10, in clasa y, aceasta valoarea va fi decrementata cu 5 si ulterior incrementata in clasa Z cu 9. Prin urmare, v va avea valoare 14. La sfarsit se apeleaza “y.getV()” care va printa pe ecran valoarea v.

1. A) Mostenirea de clasa si suprascrierea este facuta corect in exemplu, dar nu este posibila folosirea unui camp nestatic dintr-o metoda statica si asta va genera o eroare de compilare.

Se poate rezolva problema facand campul X static in felul urmator: “private static X a = new Y()”. Cu aceasta modificare in program, nu se va mai genera eroare de compilare pe linia marcata cu “1”.

B)

Da, este vorba despre o eroare de compilare, chiar daca B este o superclasa al lui A. Nu se poate face aceasta asignare deorece daca s-ar permite asta, in container-ul Obj s-ar putea pune orice instante al lui A sau o subclasa al lui A diferita de B, prin urmare, ar aparea o eroare de abia la runtime.

Daca doresc sa-mi ruleze asignarea din cerinta, ar trebui sa folosesc windcard in felul urmator:

Obj<? extends A> = new Obj<B>();

Dar in acest mod de scriere

1. A)Pe linia marcata cu “1” se va tipari valoarea -13, deoarece:

La prima iteratie a buclei for din metoda main, apelul metodei “make” se termina cu exceptie, pentru ca x are valoarea 0, iar in blocul finally x devine 1.

La a doua iteratie a buclei for, metoda “make” a clasei b va retura valorea -7 si nu se arunca vreo exceptie. In finally, x devine -6.

In ultima iteratie a buclei for, “make” va returna -14 si in finally x devine -13. Aceasta va fi afisata

B) Daca s-ar sterge clauza throws, intr-adevar se va genera o eroare de compilare, pentru ca metoda “make” din clasa B suprascrie metoda “make” din clasa A. Nicio metoda care suprascrie alta metoda, nu are voie sa arunce alte exceptii, decat cele care sunt aruncate de metoda initiala, care va fi suprascrisa. Se mai poate ca exceptia care suprascrie sa arunce o exceptie derivata din exceptia pe care o arunca metoda initiala. In ca un se respecta aceasta regula, se genereaza o eroare de compilare. Pana la urma, este logic sa fie asa, altfel la rulare, o metoda ar putea sa arunce exceptii pe care un le-a declarat, acest lucru ar duce la erori greu de depanat.

6. A)

Da, este posibila accesarea campului x pentru clasa C extinde clasa B care mosteneste campul protected x. Mentionez ca accesarea campului x se poate face doar pe this, si pe niciun alt obiect, creat in interior, deoarece clasa C se afla intr.-un pachet diferit si se respecta conditia impusa de protected ca un camp de acest fel sa poata fi accesat doar pe this. Daca C ar fi fost in acelasi pachet, s-ar fi putut face accesarea elementului de oriunde, cu toate ca acest lucru incalca principiul teoretic al modificatorului de acces, protected.

B) Daca nu se modifica nimic in codul sursa prezentat, chiar cele doua clase un pot aparea in aceeasi unitate de compilare. Motivul este simplu, daca ar fi implementate in aceeasi unitate, cel putin o clasa nu ar putea avea specificatorul de acces public, prin urmare, un ar putea fi accesat din alt pachet. In implementarea clasei C am nevoie si de clasa B pentru a extinde clasa C si de clasa A, a carei instanta va veni ca parametru metodei “doSomething”. Prin urmare, ambele clase, A si B, trebuie sa fie declarate public, ceea ce un ar fi posibil daca ar aparea in aceeasi unitate de compilare.