1. A)

I orice clasa la crearea unui obiect nou, prima data se executa apelarea superconstructorului, apoi asignarile care apar in afara constructorului in ordine textuala. Deci, cand se apeleaza “new B()”, se apeleaza superconstructorul, apoi x devine 0, iar apoi se executa corpul constructorului, care in cazul de fata nu exista. Deci, x ar ramane pe 0, daca variabila locala b ar primi o instanta a clasei B, dar aceasta neexistand un se poate vorbi despre valoarea lui x din clasa B spre care refera referinta b, deoarece obiectul un a fost creat, deci variabila x un exista.

B)

Se poate pune specificatorul de acces default in loc de protected de forma: “int x;”, in acest fel, campul x se poate accesa de oriunde din interiorul pachetului, dar nu si din afara in vreo forma sau alta.

1. A) In blocul try se intra in bucla for si se executa prima iteratie si variabila a va avea valoarea 1, la a doua iteratie apelarea metodei “exec” care se termina cu exceptie si se iese din blocul try. I finally se mai incrementeaza cu 1 variabila a si va deveni 2.

Pe linia marcata cu “1” se va tipari 2.

B) Clasa “Except” extinde “RuntimeException” , deci este o exceptie neverificata. Prin urmare un este neaparata nevoie ca la apelarea unei metode care poate arunca o astfel de exceptie sa apara intr.-un bloc try-catch. Bineinteles, netratarea exceptiei poate duce la erori la runtime, dar codul ar fi compilabil in orice caz.

1. A)

Pe linia marcata cu “1” se va tipari valoarea 36, deoarece:

Referinta p va primi o instante “Pair(12, 11)”. Mai jos referinta s va pointa si ea catre acest obiect dupa executarea liniei “s = p”. La executarea metodei “call” pe prima linia obiectul referit de p si s este modificat in: “Pair(36, 35)” si acesta un se mai modifica pana la incheierea programului.

Prin urmare, se va tipari pe ecran 36.

B) Pe linia marcata cu “2” se va tipari valoarea 23, deoarece:

Referinta q va primi la inceput o instante de forma “Pair(24, 23)”. Acest obiect un este modificat in urma executarii metodei “call” la momentul cand se executa linia: “b = new Pair(48, 47” deoarece b este o referinta separata de q. Prin urmare, obiectul referit de b va fi prelucrata in continuare, neafectand referinta q si nici obiectul referit de acesta.

1. A) Mostenirea un este potrivita pentru implmentarea clasei triunghi, deoarece:

In primul rand, in afara de cele trei puncte trimite prin constructor, se pot trimite si alte puncte in plus care ar strica modelul triunghiular desenat de cele trei puncte. O solutie ar fi sa se suprascrie metoda de adaugare si acest lucru ar limita folosirea metodei din clasa de baza. Chiar si asa, un este corect sa spunem ca un triunghi este un fel de linie poligoanal (“is a”) pentru ca acest lucru ar limita facilitatile oferite de clasa de baza.

public static void main(String[] args)

{

Triunghi t = new Triunghi(1, 1, 2, 2, 3, 3);

t.adaugaUrmatorulPunct(1, 4);

//Acest lucru ar incalca concepul clasei triunghi

//Se poate rezolva cu Overriding al metodei "adaugaUrmatorulPunct",

//dar chiar si in acest caz, este gresit sa spun ca o LiniePoligonala este un fel de Triunghi ("is a")

//Implementarea corecta se face cu "has a"

}

B)

class Triunghi

{

private LiniePoligonala lp;

public Triunghi(int x1, int y1, int x2, int y2, int x3, int y3)

{

lp = new LiniePoligonala();

lp.adaugaUrmatorulPunct(x1, y1);

lp.adaugaUrmatorulPunct(x2, y2);

lp.adaugaUrmatorulPunct(x3, y3);

lp.adaugaUrmatorulPunct(x1, y1);

}

public double perimetru()

{

return lp.lungime();

}

}

1. A)

Intr-o interfata toate campurile sunt de forma “public static final”, ceea ce m-ar forta sa pun specificatorul de acces private pentru campul nume. In plus, campul fiind static, acesta va fi absolut la fel pentru toate instantele claselor carea r implementa acea interfata. Si in final, campul fiind final, aceasta un se poate modifica in vreun fel sau altul.

B) Este intr.-adevar vorba despre o eroare de compilare, deoarece in interfata un avem metoda “getTip”. Pe linia “1” un este folosit down-cast, deci acesta lipsind, se va genera eroare de compilare. O solutie pentru aceasta problema este scrierea in forma urmatoare:

1. A) Afirmatia este corecta pentru ca conceptul de legare dinámica apare la suprasciere de metode cand o metoda din subclasa are aceeasi semnatura ca si metoda din superclasa.

In cazul constructorilor, intr-adevar se executa ierarhic superconstructorii pana cand se ajunge la clasa Object, dar se stie intotdeauna exact ce superconstructor va fi apelat, acest lucru inseamna ca un se stabileste la runtime, se stabileste la compilare. Deci, un poate fi vorba de legare dinámica.

B) Daca se scrie o clasa in care se scrie o metoda statica, aceasta metoda un va puta accesa in mod direct campurile nestatice ale clasei si nici metodele nestatice. Motivul este simplu, metodele statice se pot accesa doar cu numele clasei, fara a fi nevoie de vreo instanta clara:   
“NumeClasa.metodaStatica()”, in acest moment un se poate specifica in ce clasa se afla campurile nestatice, pentru ca metoda un a fost apelata de catre un obiect.