

P1

1) Ce valoare întreagă se va tipări la linia marcată cu /* în urma executiei de mai jos?

class A{

private static int x; // x-ul e la fel pentru tot
private int y;
public A(int x1, int y1){
 x=x1;
 y=y1;
}

public void setXY(int x1, int y1){
 x=x1;
 y=y1;
}

public int getX(){
 return x;
}

public int getY(){
 return y;
}

}

class Main{

public static void m(A z, A t){

z.setXY(3,20); -> setarea x și y obiectului pe 3 și 20

z = new A(2,17); -> oici creșterea practic nu obiect nou c, care nu are și ajutor
-> singura schimbare care vorbind din asta e că x-ul este setat 2, pentru tot

public static void main(String[] args){} - programul începe de aici

A a = new A(9,50);

A b = new A(7,28);

m(a,b);

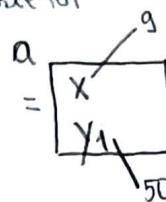
System.out.println(a.getY() + b.getX()); /*

}

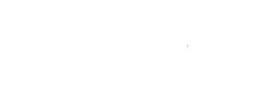
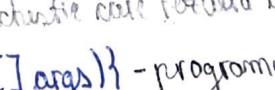
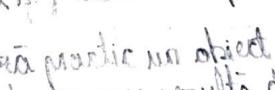
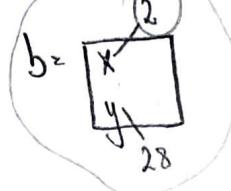
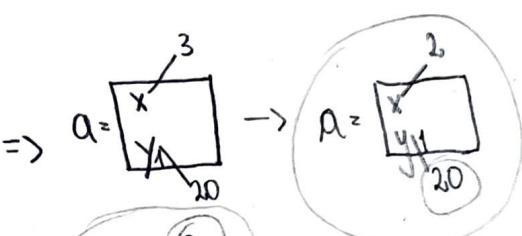
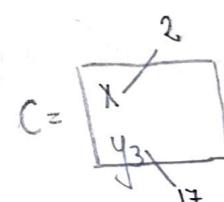
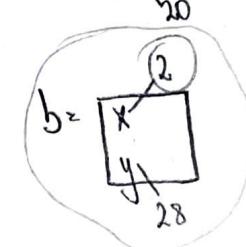
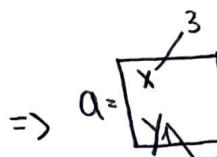
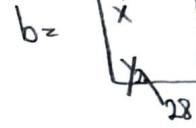
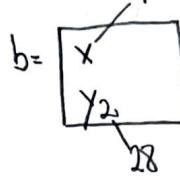
$$\Rightarrow 20 + 2 = 22$$

}

(practic ultima lui atribuire ex-x-ul nu va fi acelasi pentru tot)



{dar x-static,
ultima atribuire}



2) Ce valoare întreagă se va tipări la linia marcată cu /* în urma executiei programului?

```
class Ex extends RuntimeException {  
class Main {  
    public static void m(int x) throws Ex {  
        if (x==2) throw new Ex();  
    }  
    public static void main(String[] args) {  
        int a=2;  
        try {  
            for(int i=0; i<4; i++) {  
                m(i);  
                a=a+2;  
            }  
        } catch(Ex e) {  
            a++;  
        } finally {  
            a++;  
        }  
        System.out.println(a); /*  
    }  
}
```

$a=2$
 $i=0 \Rightarrow m(0)$
 $0 \neq 2$
 $a=a+2=2+2=4$
 $i=1 \Rightarrow m(1)$
 $1 \neq 2$
 $a=a+2=4+2=6$
 $i=2 \Rightarrow m(2)$
 $2 \neq 2 \Rightarrow \text{Exptie.}$
 $a++ \Rightarrow a=7.$
 $\text{finally: } a++ \Rightarrow a=8$

! dacă condiția din throws Ex e adevarată, nu duc la catch(Ex e)

! ce e în blocul finally se execută mereu!

3) Considerând următorul cod Java, care din fragmentele de cod enumerate mai jos vor produce o eroare când sunt inserate /*AICI*/?

abstract class A {

```
    public void m(int p) {}  
    public void r(int p) {}  
}
```

class B extends A {

```
    public void m(int p) {}  
    public void p(int p) {}  
    public void g(int p) {}  
}
```

class C extends A {

```
    public void m(int p) {}  
    public void p(int p) {}  
    public void r(int p, int t) {}  
}
```

class Main {

```
    public static void main (String [] args) {  
        A a = new B();  
        C c = new C();  
        /*AICI*/  
    }
```

SELECTAȚI UNUL SAU MAI MULTE:

- c.equals(a); → merge întotdeauna (nu are cum să dea eroare pt că tot ce face el e să returneze True F)
- c.r(i) → merge, cu toate că r(int p, int t) are doi parametrii în C. El urcă în C, vede că r are 2 param (NU-i OK), apoi urcă în A unde vede că r are 1 param, deci e OK. E OK dacă intr-unul din cele 2, cel puțin unul are 1 param. Dacă în ambele sunt 2, nu e OK.
- a.g(i) → nu merge pt că a-ului meu e de tip A, chiar dacă e new B(). În A nu suntem metodele p și q ⇒ nu merge. Dar că avem B a = new B() merge. Concluzie: nu suntem la clasa din faptă
- a.m(i) → merge pt că suntem metoda m() în clasa A. Merge și dacă n-o suntem în B pt că e suntem în A. Avem eroare dacă o suntem în B dar nu o suntem în A.

5) Considerând următorul cod Java, care din fragmentele de cod enumerate mai jos ver provoacă erori inserate /*Aici*/.

```
class A {  
    private int a=0;  
    private static int b=0;  
    public int m(){  
        return a;  
    }  
    public static int m(A x){  
        /*Aici*/  
    }  
}
```

- `return m();`: EROARE pt că m() nu e metodă statică; și și dacă ar fi `public static int m()` tot ar fi eroare pt că metoda îl returnează pe a, care nu e static. Dacă ar fi fost `public static int m() { };` și `private static int a=0` → nu ar fi fost EROARE
- `return b;`: E OK: poate returna b pt că funcția are acces doar la date statice și cum b e static ⇒ EOK
- `return x.a;`: E OK: x.a este o valoare (int) carecore; somehow, datorită parametrului (A x), x.a nu e văzut ca o valoare non-statică, e văzut ca un nr.
- `return this.a;`: EROARE: metodele staticice nu au acces la variabilele de pe care sunt apelate; nu putem apela this pe metodă statică; dacă avem `public int m(A x)`, adică fară static, era OK și putea returna `this.a`
- `return a;`: EROARE: a nu e valoare statică ⇒ nu am cum să returnez val non-statică la metodă statică

6) Ce valoare întreagă se va tipări la linia marcată cu /* în urma executiei programului de mai jos?

class A {

public int m(A p) {return 12;}

public int m(A p) {return 38;}

}

class B extends A {

public int m(B p) {return 3;}

public int m(A p) {return 7;}

}

class C extends A {

public int m(A p) {return 38;}

public int m(A p) {return 4;}

}

class D extends A {

public int m(A p) {return 1;}

public int m(D p) {return 2;}

}

class Main {

public static void main(String[] args) {

A x = new C();

A y = new D();

D z = new D();

System.out.println(x.m(z)+y.m(z)); /*

}

→ pt x: mā duc in Clasa C, mā uit la m() pt că de ola sun nerec; mā duc și in Clasa A să vad dacă sun aceeași metodă m() cu aceeași semnătură;
~dacă am în ambele aceeași metodă cu aceeași semnătură → mā duc in clasa de după new, respectiv C, și sun return-ul de acolo ⇒ x.m(z)=38

→ pt y: mā duc in Clasa D, mā uit la m(); mā duc pt-n A; metodele nu au aceeași semnătură → mā duc in clasa de dinainte de new, respectiv A ⇒ y.m(z)=38

CONCLUZIE: ~aceeași metodă cu aceeași semnătură → return din clasa de după new
~aceeași metodă, semnături diferite → return din clasa de dinainte de new

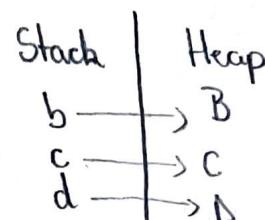
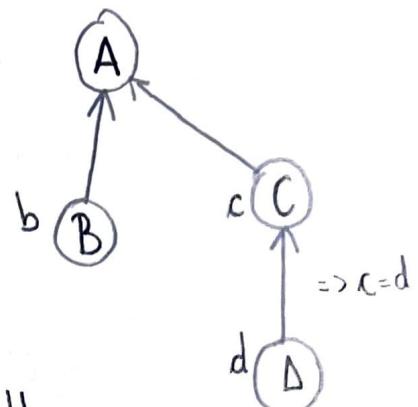
7) Considerând următorul cod Java, care din fragmentele de cod enumerate mai jos vor produce o eroare de compilare când sunt inserate aici?

```
abstract class A{}  
class B extends A{}  
class C extends A{}  
class D extends C{}  
class Main{
```

```
    public static void main(String[] args){}
```

```
        B b = new B();  
        C c = new C();  
        D d = new D();  
        /*Aici*/
```

```
}
```



$\rightarrow c = b$; EROARE: b-ul nu poate fi convertit la c;

$\rightarrow A a = new A()$; EROARE: cum A e clasa abstractă, nu am voie instantia un obiect al unei clase abstracte; dacă A nu era abstract, atunci era OK

$\rightarrow c = d$; E OK: practic conversare dintr-un tip mai mare într-un tip mai mic (partile potențial fie egale cu capitolul)

$\rightarrow A a = d$; E OK: am voie să fac partea cu A a (nu am voie să folosesc new)

$\rightarrow d = c$; EROARE: nu am voie să convertesc dintr-un tip mai mic într-unul mai mare (capitolul nu poate fi egal)

8) Considerând următorul cod Java, care din fragmentele de cod enumerate mai jos vor conduce la o eroare de EXECUȚIE când apăr la poziția indicată cu /*Aici*/?

```
class A{  
    public void m(){  
    public static void main (String [] args){  
        A [] a = new A [2];  
        /*Aici*/  
    }  
}
```

→ $a[0] = \text{new } A();$: E OK; tabloul meu are 2 elemente: $a[0]$ și $a[1]$.
 ↳ pe poziția $a[0]$ creez un obiect de tip A.

→ $a[0].m()$: EROARE: nu să pot să apeluz metoda $m()$ pe $a[0]$, trebuie creat prima oară. (prin $a[0] = \text{new } A()$ și apoi $a[0].m()$)

→ $a[2] = \text{new } A();$: NU E OK/ERROARE: tabloul meu are doar 2 elemente: $a[0]$ și $a[1]$. $a[2]$ => index out of bounds.

→ $A [] x = \text{new } A[3];$ => NULL NULL NULL
 $x[0]$ $x[1]$ $x[2]$
 $x = A;$ => $x = 2$ => NULL NULL
 $x[2] = \text{new } A();$ => EROARE: tabloul meu are doar 2 elemente: $x[0]$ și $x[1]$
 => $x[2]$ - index out of bounds.

→ $a[1] = \text{new } A();$: E OK; tabloul meu are 2 elemente: $a[0]$ și $a[1]$.
 ↳ pe poziția $a[1]$ creez un obiect de tip A.

P2

2) Considerând următorul cod Java, care din expresiile enumerate mai jos vor avea valoarea de adâncă FALSE cind sunt evaluate la poziția indicată cu /*Aici*/?

```
class A {}  
class B extends A {}  
class Main {  
    public static void main(String[] args){  
        A a = new B();  
        B b = new B();  
        Object o = new A();  
        System.out.println(/*Aici*/);  
    }  
}
```

- a.getClass -> verifică dacă clasa aparține a

- A.class -> verifică dacă A este clasa

→ A.class.equals(a.getClass()); => FALSE

→ A.class.isInstance(b) : verifică dacă obiectul referit de b (adică b) este drept clasa parinte / instanță pe clasa A
=> TRUE
~dacă clasa B nu extinde clasa A => false

→ o.getClass.equals(Object.class) => FALSE

→ B.getClass.equals(b.getClass)) => TRUE

→ B.class.isInstance(a) : verifică dacă obiectul referit de a (adică a) este drept clasa parinte / instanță pe clasa B => TRUE

4) Considerând următorul cod Java, care din fragmentele de cod enumerate mai jos vor produce o eroare de compilare când sunt inserate /*AICI*/?

```

interface I {}  

interface J {}  

interface IJ extends I, J {}  

class A implements I {}  

class B extends A implements J {}  

class C extends A {}  

class D implements IJ {}  

class Main {  

    public void doSomething(I i, J j, IJ ij){  

        A a = new A();  

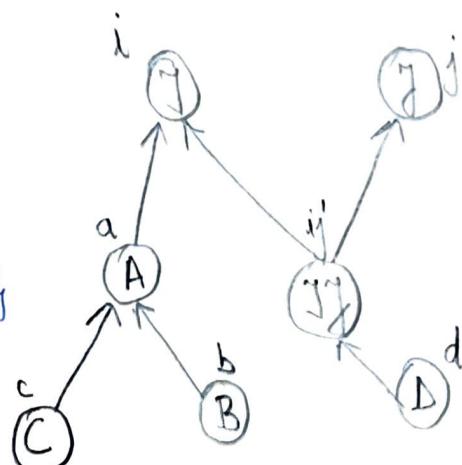
        B b = new B();  

        C c = new C();  

        D d = new D();  

        /*AICI*/  

    }
}
  
```



$\rightarrow j=c$ → EROARE: j
 $c = \text{new } C()$ ⇒ mā uit în clasa C, care e extinsă pe I;
 A implementează I

$\rightarrow b=d$ → EROARE: b $b = \text{new } B()$; class D implements IJ
 $d = \text{new } D()$; class C extends A
~~class A implements I~~

$\rightarrow i=c$ → E OK: $c = \text{new } C()$
 class C extinde A
 class A implementează I
 I

$\rightarrow ij=j$ → EROARE: $ij = \text{COPIL}$ $\leftarrow \text{COPIL} \neq \text{PĂRINTE}$
 $j = \text{PARINTE}$

$\rightarrow i=ij$ → E OK: $i = \text{PĂRINTE}$ $\leftarrow \text{PĂRINTE} \subset \text{COPIL}$
 $ij = \text{COPIL}$

5) Considerând următorul cod Java, care din fragmentele de cod enumerate mai jos vor produce o eroare de compilare?

```
interface I{  
    public void g();  
}  
  
interface J{  
    public void h();  
}  
  
abstract class A implements I{  
    public void m();  
}  
  
class B extends A implements J{  
    public void p();  
    public void q();  
    public void h();  
}  
  
class Main{  
    public static void main(String[] args){  
        J j = new B();  
        A a = new B();  
        /*Aici*/  
    }  
}
```

→ j.g(): E OK: suntem în metoda g() în clasa B

→ a.p(): EROARE: suntem în "A a = new B();"

→ clasa A este abstractă, iar în ea nu există nici metodele, și nici în interfața I

→ dacă clasa A nu era abstractă, putem să căutăm în B unde găsim

→ a.h(): EROARE

→ j.m(): EROARE

→ a.g(): E OK: → deoarece clasa A e abstractă, nu suntem în B după metoda, însă pot să căutăm în interfață, unde o găsim

6) Ce valoare întreagă se va tipări?

```
class Ex1 extends Exception{}  
class Ex2 extends Ex1{}  
class Main {  
    public static void m(int x) throws Ex1 {  
        if (x == 1) throw new Ex2();  
    }  
    public static void main(String[] args) {  
        int K=0;  
        int a=1;  
        while (K<2){  
            try{  
                K++;  
                m(K);  
                a++;  
            } catch(Ex2 e){  
                a+=2;  
            } catch(Ex1){  
                a+=3;  
            } finally{  
                a++;  
            }  
        }  
        System.out.println(a);  
    }  
}
```

- pornesc de aici

K=0

a=1

K=0 \Rightarrow mā bag pe while

K++ \Rightarrow K=1

m(1) \Rightarrow x==1 \Rightarrow Ex(2) \Rightarrow a=a+2=3

Yes pe finally \Rightarrow a=3+1=4

Revin la a2:

K=1 < 2 \Rightarrow mā bag pe while

K++ \Rightarrow K=2

m(2) \rightarrow nu am nicio excepție

a++ \Rightarrow a=5

Yes pe finally \Rightarrow a=6

7) Ce valoare întreagă se va tipări?

class A {

 private int x;

 public A(int x) { this.x = x; }

 public void setX(int x) { this.x = x; }

 public int getX() { return x; }

}

class B {

 private A y;

 public B(A y) { this.y = y; }

 public void setY(A y) { this.y = y; }

 public A getY() { return y; }

}

class Main {

 public static void m(B z, B t) {

 A w = new A(4);

 z.setY(w);

 w.setX(7);

 t = new B(new A(3));

 }

 B a = new B(new A(35));

 B b = new B(new A(33));

 m(a, b);

 System.out.println(a.getY().getX() + b.getY().getX());

}

B a = new B(new A(35)) \Rightarrow mădure în B, mă uit la constructor; văd ca parametru
A y \Rightarrow mădure în A, mă uit la constructor \Rightarrow $x_a = 35$

B b = new B(new A(33)) \Rightarrow \dots ($x_b = 33$)

m(a, b)

" "

A w = new A(4) \Rightarrow mădure în A, la constructor $\Rightarrow x_w = 4$

z.setY(w) \Rightarrow mădure în B, mă uit la setY \Rightarrow ~~x_z = x_w~~ $x_z = x_w = 4$

w.setX(7) \Rightarrow $x_w = 7 \Rightarrow x_z = 7 = x_a$

t = new B(new A(3)) \rightarrow nu vă se face schimbarea \Rightarrow rămâne ultimul x_b

$\Rightarrow 7 + 33 = 40$

8) Care din fragmentele de mai jos vor produce o eroare de compilare?

class A}

public A m(int a)

{ return new A();

}

class B extends A}

public void m(A a)

return;

}

/* AICI */

}

① public int m(A a){ } EROARE: metodele au aceeasi semnatură (acelasi nume și acelasi parametrii, tipul returnat nu contează); nu am voie să am 2 metode cu aceeasi semnatură într-o clasa.

→ public int m(B a){ } E OK: semnatura difera

}

→ public A m(B a){ } E OK: A e clasa parinte, deci merge

}

② public A m(int a) throws Exception{ } EROARE:
throws new Exception("Message");
} ~ clasa B extinde clasa A; in clasa A am
o singura metodă cu acelasi semnatură, iară
nu pot face override în B; Singura problemă
este că nu am voie să exceptii

→ public A m(int p){ } E OK

}

9)

```
class A {
    protected int a;
    public A (int a) {
        this.a = a;
    }
}
```

```
class B extends A {
    private int B;
    /*Aici*/
}
```

\rightarrow public B(int a, int b) {

```
    super(a);
    this.b = b;
}
```

} E OK (practic cu un simplu constructor)

(\Rightarrow) public B(int a, int b) {

```
    this.a = a;
    this.b = b;
}
```

} ERORARE (nu am super(a))

(\Rightarrow) public void m1(int a, int b) {

```
    super(a);
    this.b = b;
}
```

} ERORARE: nu pot "apela" doar in constructor

\rightarrow public B(int b) {

```
    super(0);
    this.b = b;
}
```

} E OK

\rightarrow public B(int a, int b) {

```
    this.b = b;
    super(a);
}
```

} ERORARE: "super" trebuie sa fie primul lucru in constructor