Problem 1

Parent class `P`

- Fields: `int a = 10; int b = 20; int c = 30;`
- Methods:
  - `void m1()` → `System.out.println("P.m1 called");`
  - `void m2()` → `System.out.println("P.m2 called");`

Child class `C` extends `P`

- Fields: `int a = 100; int d = 40; int e = 50;`
- Methods:
  - `void m2()` → `System.out.println("C.m2 called");`
  - `void m3()` → `System.out.println("C.m3 called");`

Create & test (exact):

1. `P p1 = new P();` — print `p1.a`, `p1.b`, `p1.c`; call `p1.m1();` and `p1.m2();`
2. `P p2 = new C();` — print `p2.a`, `p2.b`, `p2.c`; call `p2.m1();`, `p2.m2();`, `p2.m3();`
3. `C c1 = new C();` — print `c1.a`, `c1.b`, `c1.c`, `c1.d`, `c1.e`; call `c1.m1();`, `c1.m2();`, `c1.m3();`

Record outputs and any compile-time errors (e.g., calls not allowed by reference type).

Problem 2

Parent class `Alpha`

- Fields: `int x = 1; int y = 2;`
- Methods:
  - `void show()` → `System.out.println("Alpha.show called");`
  - `void calc()` → `System.out.println("Alpha.calc called");`

Child class `Beta` extends `Alpha`

- Fields: `int x = 10; int z = 3;`
- Methods:
  - `void calc()` → `System.out.println("Beta.calc called");`
  - `void extra()` → `System.out.println("Beta.extra called");`

Create & test (exact):

1. `Alpha a1 = new Alpha();` — print `a1.x`, `a1.y`; call `a1.show();`, `a1.calc();`
2. `Alpha a2 = new Beta();` — print `a2.x`, `a2.y`; call `a2.show();`, `a2.calc();`, `a2.extra();`
3. `Beta b1 = new Beta();` — print `b1.x`, `b1.y`, `b1.z`; call `b1.show();`, `b1.calc();`, `b1.extra();`

Record outputs & errors.

Problem 3

Parent class `Base1`

- Fields: `int p = 5; int q = 6;`
- Methods:
  - `void hello()` → `System.out.println("Base1.hello called");`
  - `void bye()` → `System.out.println("Base1.bye called");`

Child class `Child1` extends `Base1`

- Fields: `int p = 50; int r = 7;`
- Methods:
  - `void bye()` → `System.out.println("Child1.bye called");`

- `void childMethod()` → `System.out.println("Child1.childMethod called");`

Create & test (exact):

1. `Base1 bp = new Base1();` — print `bp.p`, `bp.q`; call `bp.hello();`, `bp.bye();`
2. `Base1 bp2 = new Child1();` — print `bp2.p`, `bp2.q`; call `bp2.hello();`, `bp2.bye();`, `bp2.childMethod();`
3. `Child1 ch = new Child1();` — print `ch.p`, `ch.q`, `ch.r`; call `ch.hello();`, `ch.bye();`, `ch.childMethod();`

Record outputs & errors.

Problem 4

Class `A` (top parent)

- Fields: `int a = 1; int b = 2;`
- Methods:
  - `void mA()` → `System.out.println("A.mA called");`
  - `void common()` → `System.out.println("A.common called");`

Class `B` extends `A`

- Fields: `int b = 20; int c = 3;`
- Methods:
  - `void common()` → `System.out.println("B.common called");`
  - `void mB()` → `System.out.println("B.mB called");`

Class `C` extends `B`

- Fields: `int c = 300; int d = 4;`
- Methods:
  - `void common()` → `System.out.println("C.common called");`
  - `void mC()` → `System.out.println("C.mC called");`

Create & test (exact):

1. `A obj1 = new A();` — print `obj1.a`, `obj1.b`; call `obj1.mA();`, `obj1.common();`
2. `A obj2 = new C();` — print `obj2.a`, `obj2.b`; call `obj2.mA();`, `obj2.common();`, `obj2.mC();`
3. `C obj3 = new C();` — print `obj3.a`, `obj3.b`, `obj3.c`, `obj3.d`; call `obj3.mA();`, `obj3.mB();`, `obj3.mC();`, `obj3.common();`

Record outputs & note compile errors when a reference does not expose a child-only member.

Problem 5

Class `Top`

- Fields: `int v = 10;`
- Methods: `void topMethod()` → `System.out.println("Top.topMethod called");`

Class `Mid1` extends `Top`

- Fields: `int v = 100;`
- Methods: `void midMethod()` → `System.out.println("Mid1.midMethod called");`, `void topMethod()` → `System.out.println("Mid1.topMethod called");`

Class `Bottom` extends `Mid1`

- Fields: `int v = 1000; int z = 9;`
- Methods: `void bottomMethod()` → `System.out.println("Bottom.bottomMethod called");`, `void topMethod()` → `System.out.println("Bottom.topMethod called");`

Create & test (exact):

1. `Top t1 = new Top();` — print `t1.v`; call `t1.topMethod();`
2. `Top t2 = new Bottom();` — print `t2.v`; call `t2.topMethod();`, `t2.bottomMethod();`

3. `Bottom b = new Bottom();` — print `b.v`; call `b.topMethod();`, `b.midMethod();`, `b.bottomMethod();`

Record outputs & compiler messages if any.

Problem 6

Class `Alpha1`

- Fields: `int f = 2;`
- Methods: `void one()` → `System.out.println("Alpha1.one called");`

Class `Beta1` extends `Alpha1`

- Fields: `int f = 20;`
- Methods: `void two()` → `System.out.println("Beta1.two called");`, `void one()` → `System.out.println("Beta1.one called");`

Class `Gamma1` extends `Beta1`

- Fields: `int f = 200; int g = 7;`
- Methods: `void three()` → `System.out.println("Gamma1.three called");`, `void one()` → `System.out.println("Gamma1.one called");`

Create & test (exact):

1. `Alpha1 a = new Gamma1();` — print `a.f`; call `a.one();`, `a.three();`
2. `Beta1 b = new Gamma1();` — print `b.f`; call `b.one();`, `b.two();`, `b.three();`
3. `Gamma1 g = new Gamma1();` — print `g.f`, `g.g`; call `g.one();`, `g.two();`, `g.three();`

Record outputs & errors.

Problem 7

Class `L1`

- Fields: `int a = 1;`
- Methods: `void s1()` → `System.out.println("L1.s1 called");`

Class `L2` extends `L1`

- Fields: `int a = 2;`
- Methods: `void s2()` → `System.out.println("L2.s2 called");`, `void s1()` → `System.out.println("L2.s1 called");`

Class `L3` extends `L2`

- Fields: `int a = 3;`
- Methods: `void s3()` → `System.out.println("L3.s3 called");`, `void s1()` → `System.out.println("L3.s1 called");`

Class `L4` extends `L3`

- Fields: `int a = 4; int extra = 99;`
- Methods: `void s4()` → `System.out.println("L4.s4 called");`, `void s1()` → `System.out.println("L4.s1 called");`

Create & test (exact):

1. `L1 x1 = new L1();` — print `x1.a`; call `x1.s1();`
2. `L1 x2 = new L4();` — print `x2.a`; call `x2.s1();`, `x2.s4();`
3. `L4 x3 = new L4();` — print `x3.a`, `x3.extra`; call `x3.s1();`, `x3.s2();`, `x3.s3();`, `x3.s4();`

Record outputs & errors.

Problem 8

Chain: `Root` → `Node1` → `Node2` → `Leaf`

Root

- Fields: `int val = 100;`
- Methods: `void rootMsg()` → `System.out.println("Root.rootMsg called");`

Node1 extends Root

- Fields: `int val = 200;`
- Methods: `void node1Msg()` → `System.out.println("Node1.node1Msg called");`, `void rootMsg()` → `System.out.println("Node1.rootMsg called");`

Node2 extends Node1

- Fields: `int val = 300;`
- Methods: `void node2Msg()` → `System.out.println("Node2.node2Msg called");`, `void rootMsg()` → `System.out.println("Node2.rootMsg called");`

Leaf extends Node2

- Fields: `int val = 400; int leafOnly = 7;`
- Methods: `void leafMsg()` → `System.out.println("Leaf.leafMsg called");`, `void rootMsg()` → `System.out.println("Leaf.rootMsg called");`

Create & test (exact):

1. `Root r = new Leaf();` — print `r.val`; call `r.rootMsg();`, `r.leafMsg();`
2. `Node1 n1 = new Leaf();` — print `n1.val`; call `n1.rootMsg();`, `n1.node2Msg();`
3. `Leaf lf = new Leaf();` — print `lf.val`, `lf.leafOnly`; call `lf.rootMsg();`, `lf.node1Msg();`, `lf.node2Msg();`, `lf.leafMsg();`

Record outputs & any compiler errors.

Problem 9

Chain of four with different names (`One` → `Two` → `Three` → `Four`)

One

- Fields: `int f = 1;`
- Methods: `void A()` → `System.out.println("One.A called");`

Two extends One

- Fields: `int f = 2;`
- Methods: `void B()` → `System.out.println("Two.B called");`, `void A()` → `System.out.println("Two.A called");`

Three extends Two

- Fields: `int f = 3;`
- Methods: `void C()` → `System.out.println("Three.C called");`, `void A()` → `System.out.println("Three.A called");`

Four extends Three

- Fields: `int f = 4; int extra = 44;`
- Methods: `void D()` → `System.out.println("Four.D called");`, `void A()` → `System.out.println("Four.A called");`

Create & test (exact):

1. `One o = new Four();` — print `o.f`; call `o.A();`, `o.D();`
2. `Three t = new Four();` — print `t.f`; call `t.A();`, `t.C();`, `t.D();`
3. `Four fo = new Four();` — print `fo.f`, `fo.extra`; call `fo.A();`, `fo.B();`, `fo.C();`, `fo.D();`

Record outputs & errors.

Problem 10

One

- Fields: `int val = 10;`
- Methods: `void step()` → `System.out.println("One.step called");`

Two extends One

- Fields: `int val = 20;`
- Methods: `void step()` → `System.out.println("Two.step called");`, `void twoOnly()` → `System.out.println("Two.twoOnly called");`

Three extends Two

- Fields: `int val = 30;`
- Methods: `void step()` → `System.out.println("Three.step called");`, `void threeOnly()` → `System.out.println("Three.threeOnly called");`

Four extends Three

- Fields: `int val = 40;`
- Methods: `void step()` → `System.out.println("Four.step called");`, `void fourOnly()` → `System.out.println("Four.fourOnly called");`

Five extends Four

- Fields: `int val = 50; int last = 999;`
- Methods: `void step()` → `System.out.println("Five.step called");`, `void fiveOnly()` → `System.out.println("Five.fiveOnly called");`

Create & test (exact):

1. `One o1 = new Five();` — print `o1.val`; call `o1.step();`, `o1.fiveOnly();`
2. `Three t3 = new Five();` — print `t3.val`; call `t3.step();`, `t3.fourOnly();`, `t3.fiveOnly();`

3. `Five f5 = new Five();` — print `f5.val`, `f5.last`; call `f5.step();`, `f5.twoOnly();`, `f5.threeOnly();`, `f5.fourOnly();`, `f5.fiveOnly();`

Record outputs & any compiler errors.
