

Static & Dynamic Types

All variables in Java have a static type. This is the type we declared the variable as. Can only declare a variable once → static type doesn't change.

★ Animal a;

OR

★ Animal a = ... ;

~~x = 5~~
int x = 5;

When we assign a variable (by using =), this affects its dynamic type.

The dynamic type is what kind of object the variable actually points to. can reassign many times → dynamic type can change!



★ ^{Animal box} a = new Dog(); /* dynamic type of a is Dog */

★ a = new Cat(); /* dynamic - cat */

Think of the compiler as a very cautious "proofreader"
It keeps an eye out for potential mistakes.

* Static type = compile-time type

* dynamic type = run-time type

The compiler must "proofread" with limited info: only knows static type!

It errs on the cautious side ... it sometimes thinks there is a mistake even if at runtime there would be no error.

	Static	Dynamic
a	Animal	Dog

* Animal a = new Dog();

* a. bark(); → COMPILER ERROR
↓
Animal

In cases like this, we have to promise the compiler that at run-time, a will be a dog \Rightarrow CASTING

((Dog) a).bark();

If we "break the promise", then we will get an error at run-time instead.

*Animal a = new Animal(); a ^S Animal ^D Animal

((Dog) a).bark(); (Dog) a

↪ classCastException

Important Note:

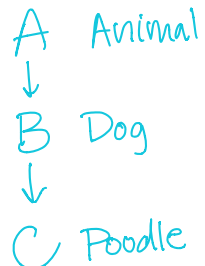
Casting is simply a promise to the compiler! Casting does not change the type (neither static nor dynamic) of an object!

3 An Exercise in Inheritance Misery *Extra*

3.1 Cross out any lines that cause compile-time errors or cascading errors (failures that occur because of an error that happened earlier in the program), and put an X through runtime errors (if any). Don't just limit your search to main, there could be errors in classes A,B,C. What does D.main output after removing these lines?

```

1  class A {
2      public int x = 5; ✓
3      public void m1() {System.out.println("Am1-> " + x);} ✓
4      public void m2() {System.out.println("Am2-> " + this.x);} ✓
5      public void update() {x = 99;} ✓
6  } subclass extends superclass
7  class B extends A {
8      public void m2() {System.out.println("Bm2-> " + x);} override
9      public void m2(int y) {System.out.println("Bm2y-> " + y);} overload
10     public void m3() {System.out.println("Bm3-> " + "called");} ✓
11 }
12 class C extends B {
13     public int y = x + 1; → y=6
14     public void m2() {System.out.println("Cm2-> " + super.x);} override
15     public void m4() {System.out.println("Cm4-> " + super.super.x);}
16     public void m5() {System.out.println("Cm5-> " + y);} ✓
17 }
18 class D {
19     public static void main (String[] args) {
20         B a0 = new A(); compile error
21         a0.m1();
22         a0.m2(16);
23         A b0 = new B();
24         System.out.println(b0.x); 5
25         b0.m1(); Am1 → 5
26         b0.m2(); Bm2 → 5
27         b0.m2(61); compile error
28         B b1 = new B();
29         b1.m2(61); Bm2y → 61
30         b1.m3(); Bm3 → called
31         A c0 = new C();
32         c0.m2(); Cm2 → 5
33         C c1 = (A) new C(); compile error
34         A a1 = (A) c0; ✓
35         C c2 = (C) a1; ✓
36         c2.m3(); Bm3 → called
37         c2.m4(); cascading
38         c2.m5(); Cm5 → 6
39         ((C) c0).m3(); Bm3 → called
40         (c) c0.m3(); compile error
41         b0.update();
42         b0.m1(); Am1 → 99
43     }
44 }
    
```



	Static	Dynamic	x	y
b0	A	B	99	—
b1	B	B	5	—
c0	A	C	5	6
a1	A	C	5	6
c2	C	C	5	6

