Overriding and Overloading Method Call: Decision Tree

```
Method call: obj.method(arg);

Step 1: Look at the _________ TYPE of obj
        (the type of the reference, not the new object).

Step 2: Compiler picks a method signature (overload resolution)
    - Based only on the _______ TYPE of obj
      + the ________ TYPE of arg.
    - Chooses the "most specific applicable method" (pick itself or closest ancestor)

Step 3: At RUNTIME, Java checks if that chosen method
        is ________ in the ________ TYPE of obj.
    - If yes → run the overriding version.
    - If no  → run the inherited version.
```

Overriding and Overloading Method Call Decision Tree with Type Casting (ULTIMATE!!)

```
Declaration and instantiation:
S1 obj = new D1(); // obj's original static type is S1 and dynamic type is D1.
S2 arg = new D2(); // arg's original static type is S2 and dynamic type is D2.

Method call: ((T1)obj).method((T2)arg);


((T1)obj).method((T2)arg);

1. At compile-time
Check (type conversion legality)
       Check: S1 -> T1 either ___ or _____ the inheritance tree?
       Check: S2 -> T2 either ___ or _____ the inheritance tree?
   └─If invalid, compile error "_____________ types"

Apply static type conversion
   ├─ obj is treated as type ____   (obj's static type = T1)
   └─ arg is treated as type ____   (obj's static type = T2)

Check existence of method: does T1.method(T2') exists in class T1?
(T2': T2 or most specific applicable type closest to T2.)
   ├─ if method itself does not exist, compile error "_________________"
   └─ if such T2' does not exist, compile error "_________________"

Picks method signature with most specific applicable overload based on T2 from T1
T1.method(T2')

2. At runtime:
Check: Allow type conversion only if S1 -> T1 is ___ the inheritance tree.
Check: Allow type conversion only if S2 -> T2 is ___ the inheritance tree.
If not, runtime ______________Exception is thrown.

Honors the method choice made in Step 1: T1.method(T2')
   └─ Call goes to dynamic type of obj (D1)
       ├─ If method is overridden → run dynamic type's version: D1.method(T2')
       └─ Else → run inherited version.
```

Example 0

```
class Parent {
    void method(C c){ … } //C is a subclass of B
}

class Child extends Parent {
    void method(A a){ … }
    void method(B b){ … } //B is a subclass of A
}

Main {
    Parent p = new Parent();
    B b = new B();
    p.method(b); //what happens?
}
```

Example 1 (Credit: Akaash)

```
class P {
  method1 (A a) { some implementation }
}

class C extends P {
  method1 (A a) { some implementation }
  method1 (B b) { some implementation }
}

class A { some implementation }
class B extends A { some implementation }

main {
  P somePC = new C;
  A someAB = new B;
  B someBB = new B;

  somePC.method1(someAB); //what happens?
  somePC.method1(someBB); //what happens?
}
```

Example 2 (Ed Post)

```
main {
    P p2 = new C(); //P is C's parent
    A a1 = new A();
    B b1 = new B();
    A a2 = new B(); //A is B's parent

    p2.method1(a1); //which version?
    p2.method1(b1); //which version?
    p2.method1(a2); //which version?
}
```