Java Casting

Table of contents

Introduced in 2026-03-12-object-oriented-programming-ii[1] (Lecture, Week 3). See also java-primitive-and-reference-types[2].

Casting converts a variable from one type to another.

Primitive casting

- **Widening** — converting a smaller type to a larger one: `byte -> short -> char -> int -> long -> float -> double`.
- **Narrowing** — converting a larger type to a smaller one: `double -> float -> long -> int -> char -> short -> byte`.

Reference casting

- **Upcasting** (implicit) — casting a subclass reference to a superclass reference.
- **Downcasting** (explicit) — casting a superclass reference back to a subclass reference.

Given `Object -> Animal -> {Mammal -> {Dog, Cat}, Bird -> Chicken}`, upcasting moves up this hierarchy, downcasting moves down it.

[1] courses/csse2002/2026-03-12-object-oriented-programming-ii.pdf
[2] courses/csse2002/java-primitive-and-reference-types.pdf

Upcasting

Always safe, done implicitly:

```
public class Animal {
    public void makeSound() { System.out.println("Animal makes a sound"); }
}
public class Cat extends Animal {
    @Override public void makeSound() { System.out.println("Cat meows"); }
}

Animal myAnimal = new Cat(); // upcasting: Cat treated as an Animal
myAnimal.makeSound();        // "Cat meows"
```

Downcasting

Must check the object is actually an instance of the target subclass with `instanceof` first, since not every `Animal` is a `Cat`:

```
public class Cat extends Animal {
    @Override public void makeSound() { System.out.println("Cat meows"); }
    public void climb() { System.out.println("I can climb trees"); }
}

Animal myAnimal = new Cat(); // upcasting
if (myAnimal instanceof Cat) {
    Cat myCat = (Cat) myAnimal; // downcasting
    myCat.climb();              // access methods specific to Cat
}
```

Every class implicitly extends `Object`

A class with no explicit `extends` clause (e.g. a plain `public class Student { ... }`) still has a parent: `Object`, the root of the Java class hierarchy.