**Overriding and overloading**

Method overriding only occurs if the method signature is *exactly* the same. Let's take a look at the following classes:

public class Bird {

public void gulgate(Bird b) { ... }

}

public class Falcon {

public void gulgate(Bird b) { ... }

public void gulgate(Falcon f) { ... }

}

Here, Falcon::gulgate(Bird) overrides Bird::gulgate(Bird) since the method signatures are *exactly* the same. Falcon::gulgate(Falcon) is an example of method overloading, where the name of the function is the same but the argument that it takes in.

*Different return types*

If we had a method in Bird with the signature "public **int** gulgate(Falcon f)" (same signature as Falcon::gulgate(Falcon) but with a different return type), Java wouldn't allow this and would complain during compilation that the return types are not compatible (void from Falcon::gulgate(Falcon) and int from Bird::gulgate(Falcon)).

*Different access modifiers*

When the access modifiers are different, something slightly different will happen (I think [this link](https://www.geeksforgeeks.org/overriding-in-java/) describes it pretty well! Ctrl-F for "Overriding and Access-Modifiers") but don't worry about this too much since we haven't covered access modifiers in depth during lecture yet.