**4. Flirbocon (12 points).** Consider the declarations below. Assume that `Falcon` extends `Bird`.

```
Bird bird = new Falcon();
Falcon falcon = (Falcon) bird;
```

Consider the following possible features for the `Bird` and `Falcon` classes. Assume that all methods are **instance methods** (not static!).

F1. The `Bird::gulgate(Bird)` method exists.[1]
F2. The `Bird::gulgate(Falcon)` method exists.
F3. The `Falcon::gulgate(Bird)` method exists.
F4. The `Falcon::gulgate(Falcon)` method exists.

The notation `Bird::gulgate(Bird)` specifies a method called `gulgate` with parameter of type `Bird` from the `Bird` class.

a) Suppose we make a call to `bird.gulgate(bird);`

Which features are sufficient **ALONE** for this call to compile? For example if feature F3 or feature F4 alone will allow this call to compile, fill in the F3 and F4 boxes.

☐ F1 ☐ F2 ☐ F3 ☐ F4 ☐ Impossible

Select a set of features such that this call executes the `Bird::gulgate(Bird)` method. For example, if having features F2 and F4 only (and not F1 or F3) would result in `Bird::gulgate(Bird)` being executed, check boxes F2 and F4 only.

☐ F1 ☐ F2 ☐ F3 ☐ F4 ☐ Impossible

Select a set of features such that this call executes the `Falcon::gulgate(Bird)` method.

☐ F1 ☐ F2 ☐ F3 ☐ F4 ☐ Impossible

b) Suppose we make a call to `falcon.gulgate(falcon);`

Which features are sufficient **ALONE** for this call to compile?

☐ F1 ☐ F2 ☐ F3 ☐ F4 ☐ Impossible

Select a set of features such that this call executes the `Bird::gulgate(Bird)` method.

☐ F1 ☐ F2 ☐ F3 ☐ F4 ☐ Impossible

Select a set of features such that this call executes the `Bird::gulgate(Falcon)` method.

☐ F1 ☐ F2 ☐ F3 ☐ F4 ☐ Impossible

Select a set of features such that this call executes the `Falcon::gulgate(Bird)` method.

☐ F1 ☐ F2 ☐ F3 ☐ F4 ☐ Impossible

Select a set of features such that this call executes the `Falcon::gulgate(Falcon)` method.

☐ F1 ☐ F2 ☐ F3 ☐ F4 ☐ Impossible

[1] In other words, the `Bird` class has a method with the signature `gulgate(Bird)`