Why prefer composition over inheritance? What trade-offs are there for each approach? When should you choose inheritance over composition?

*Prefer composition over inheritance as it is more malleable / easy to modify later, but do not use a compose-always approach.* With composition, it's easy to change behavior on the fly with Dependency Injection / Setters. Inheritance is more rigid as most languages do not allow you to derive from more than one type.. So the goose is more or less cooked once you derive from Class A.  
My acid test for the above is:

* Does TypeB want to expose the complete interface (all public methods no less) of TypeA such that TypeB can be used where TypeA is expected? Indicates **Inheritance**.

e.g. A Cessna biplane will expose the complete interface of an airplane, if not more. So that makes it fit to derive from Airplane.

* Does TypeB only want only some/part of the behavior exposed by TypeA? Indicates need for**Composition.**

e.g. A Bird may need only the fly behavior of an Airplane. In this case, it makes sense to extract it out as an interface / class / both and make it a member of both classes.