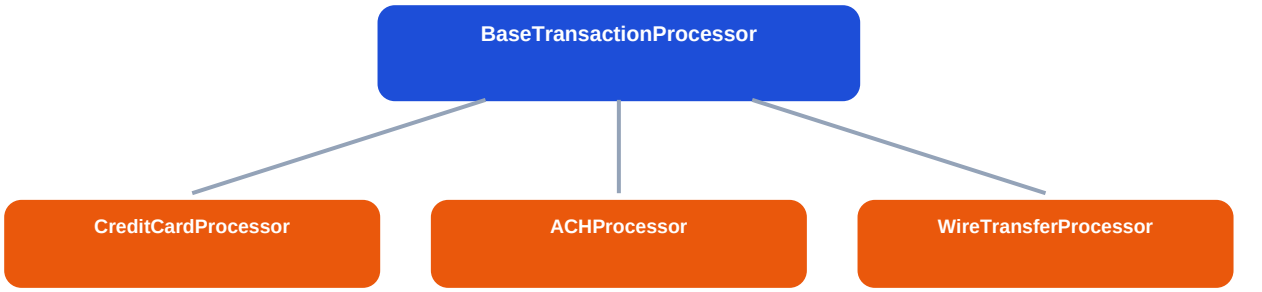
Inheritance vs. Composition Decision Checklist

Choose the right approach, and check the blast radius before changing a shared base class.

1. The Fragile Base Class Problem



All subclasses inherit the change — even ones whose tests never touched it

2. Inheritance vs. Composition

Approach	Coupling	Best For
Inheritance	Tight — depends on parent internals	A genuine, stable is-a relationship
Composition	Loose — depends only on an interface	Code reuse without a true is-a relationship

The test: can you honestly say the subclass IS a more specific version of the parent — or are you just reusing convenient methods? First case: inheritance. Second case: composition.

3. Before Changing a Shared Base Class

- ■ List every subclass that extends this base class — not just the ones you remember.
- ■ Run the FULL regression suite against every subclass, not just the feature the change was meant for.
- ■ Check whether any subclass overrides the method you're changing and calls super() at a non-obvious point.
- ■ Document the call-order contract for the method, if one doesn't already exist, before merging.
- ■ Ask whether this change could instead live in a composed helper class that subclasses opt into explicitly.
- ■ If the hierarchy is 4+ levels deep, treat any change as high-risk regardless of how small it looks.

Source: Joshua Bloch's Effective Java, a standard reference in the field, argues for favoring composition over inheritance except where a genuine, stable is-a relationship exists.