At last ... Inheritance

We have several types of movie that have different ways of answering the same question. This sounds like a job for subclasses. We can have three subclasses of movie, each of which can have its own version of charge (Figure 1.14).

![Figure 1.14. Using inheritance on movie](image)

This allows me to replace the switch statement by using polymorphism. Sadly it has one slight flaw—it doesn't work. A movie can change its classification during its lifetime. An object cannot change its class during its lifetime. There is a solution, however, the State pattern [Gang of Four]. With the State pattern the classes look like Figure 1.15.

![Figure 1.15. Using the State pattern on movie](image)

By adding the indirection we can do the subclassing from the price code object and change the price whenever we need to.

If you are familiar with the Gang of Four patterns, you may wonder, "Is this a state, or is it a strategy?" Does the price class represent an algorithm for calculating the price (in which case I prefer to call it Pricer or PricingStrategy), or does it represent a state of the movie (Star Trek X is a new release). At this stage the choice of pattern (and name) reflects how you want to think about the structure. At the moment I'm thinking about this as a state of movie. If I later decide a strategy communicates my intention better, I will refactor to do this by changing the names.