Chapter 3 : The Decorator pattern

* If behaviour is inherited by subclassing, it is set statically at compile time.
  But if behaviour is extended by composition, then it can be set dynamically at runtime.

* By composition, write new code for new features instead of altering.

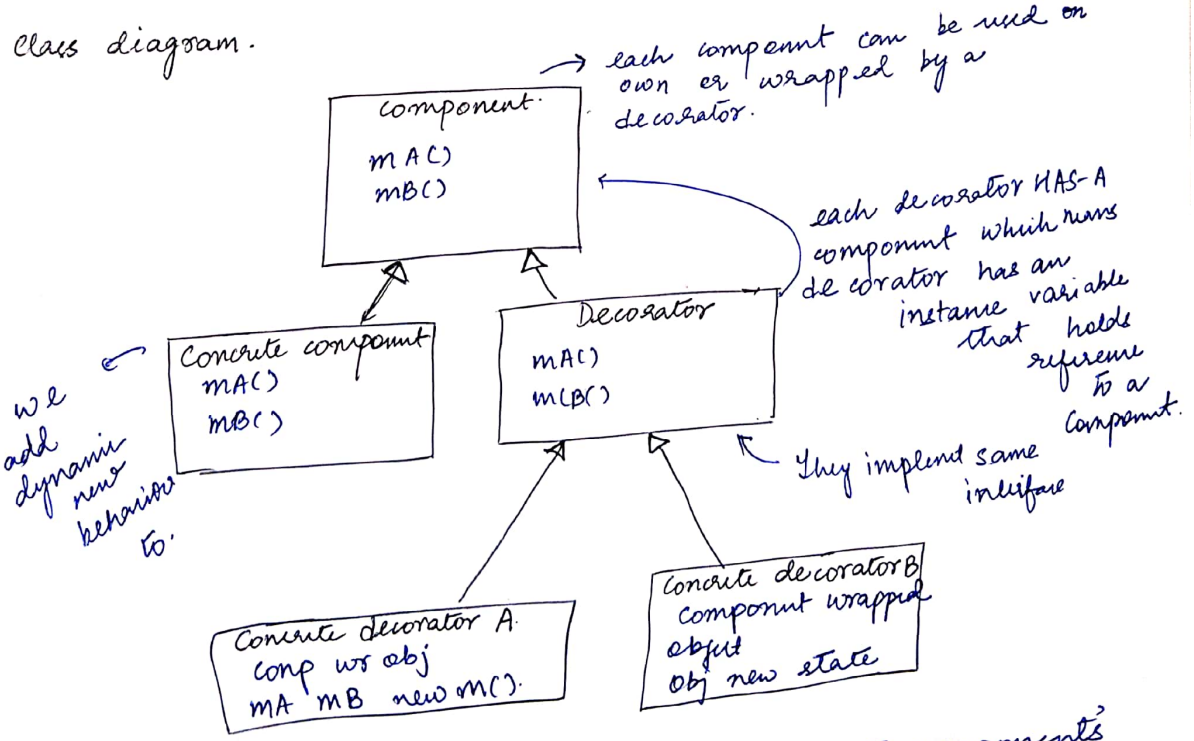
* OPEN CLOSE PRINCIPLE
  Classes should be open for extension but closed for modification.

* Decorator (as object wrappers). Add behaviour to object they wrap).

* The decorator adds its own behaviour either before and/or after delegating to the object it decorates to do the rest of the job.

* The Decorator pattern attatches additional responsibilities to an object dynamically. They provide flexible alternative to subclassing for extending functionality.

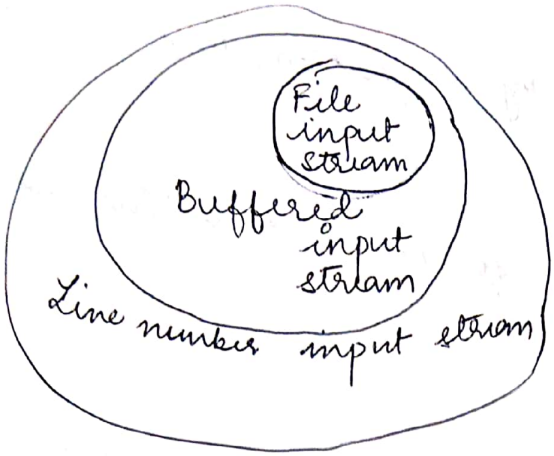
* Class diagram.



* If we have a code that relies on the concrete component's type, decorators will break that code.

* Creation of concrete component with decorator is well-encapsulated.

* Real World Decorators, Jana I/O



* Downside of decorator pattern:
→ large number of small classes that can be confusing if trying to use Decorator-based API.
→ Bad when code is dependent on specific types.
* Write your Jana I/O.
* Good about decorator:
→ Usually invent decorator & client doesn't have to know.