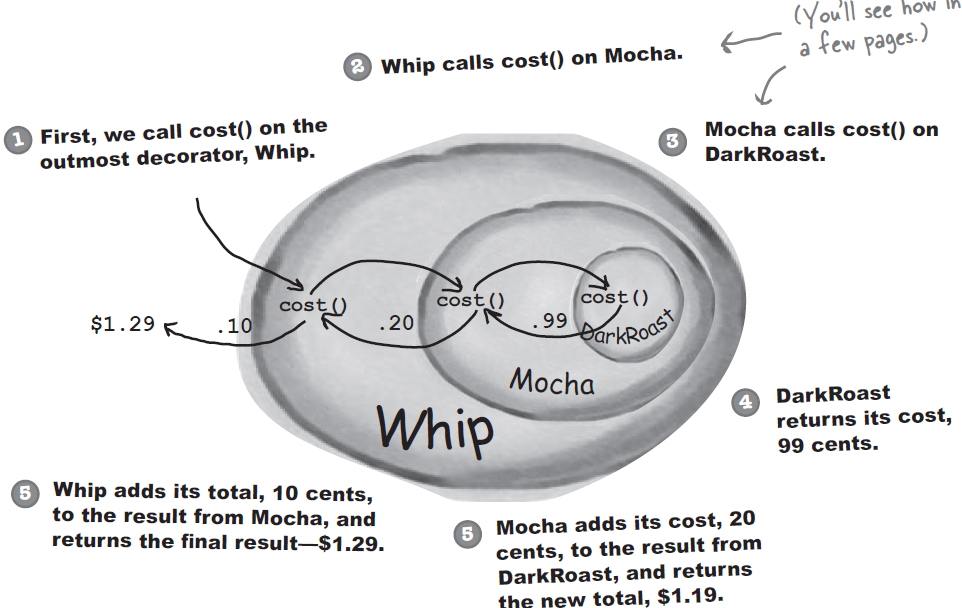
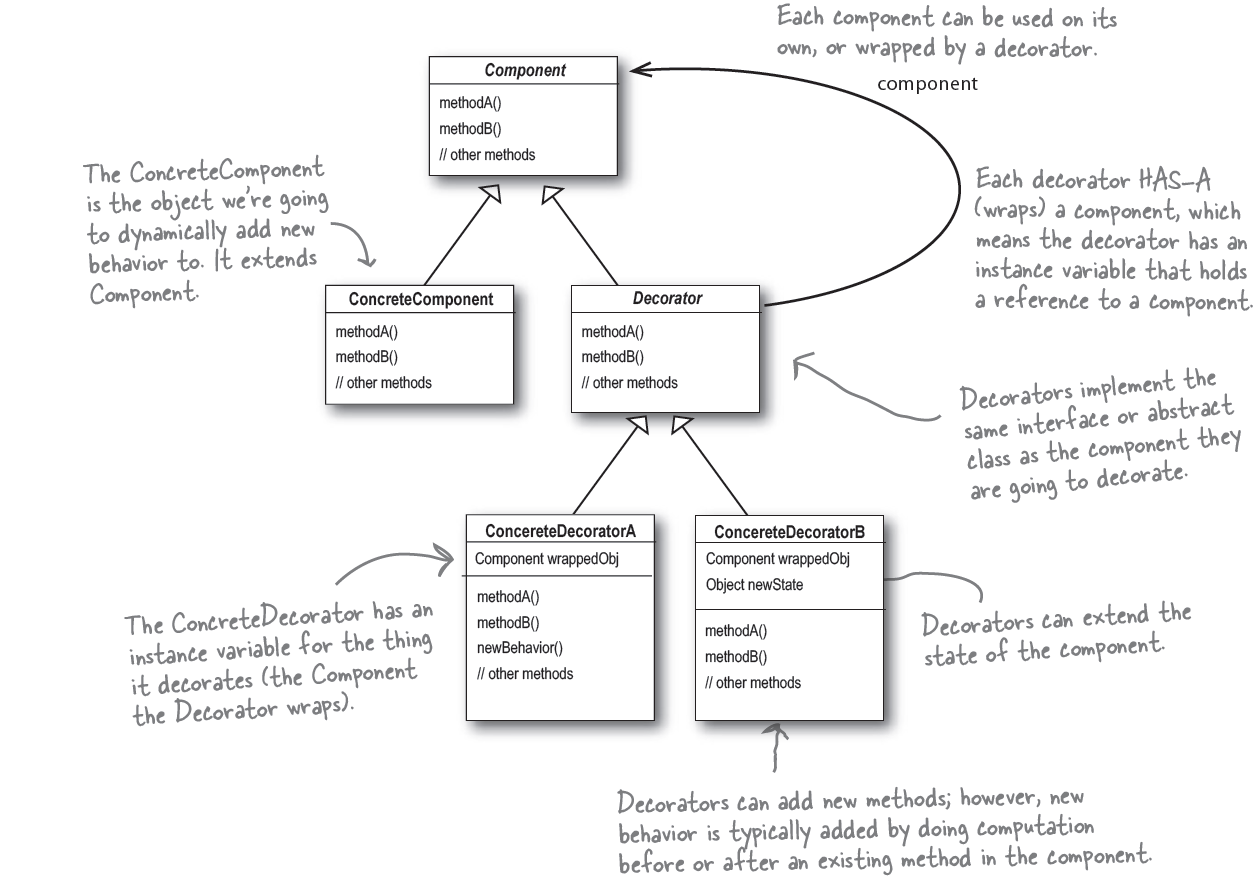
1. **Definition**: Decorator defines additional responsibilities to an object dynamically.
   1. Decorator has same subtype as the object they decorator. So, we pass the decorated object in place of original object.
   2. Object can be decorated at runtime with any number of decorators.
2. Decorator pattern allows to give new responsibility to 3rd party class object without accessing or making changes to underlying classes.
3. Classes should be open for extension but closed for modification.
   1. **Ex1**: In observer pattern, we can extend subject by adding new subscribers with modifying its code.

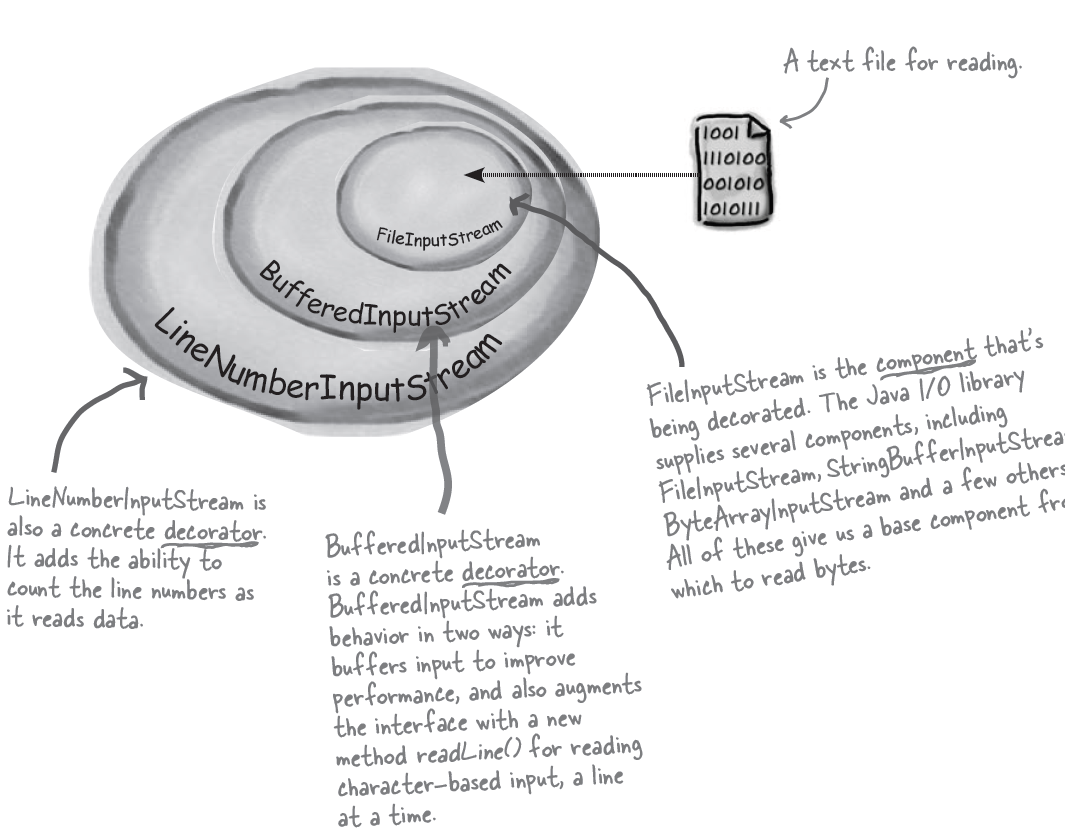


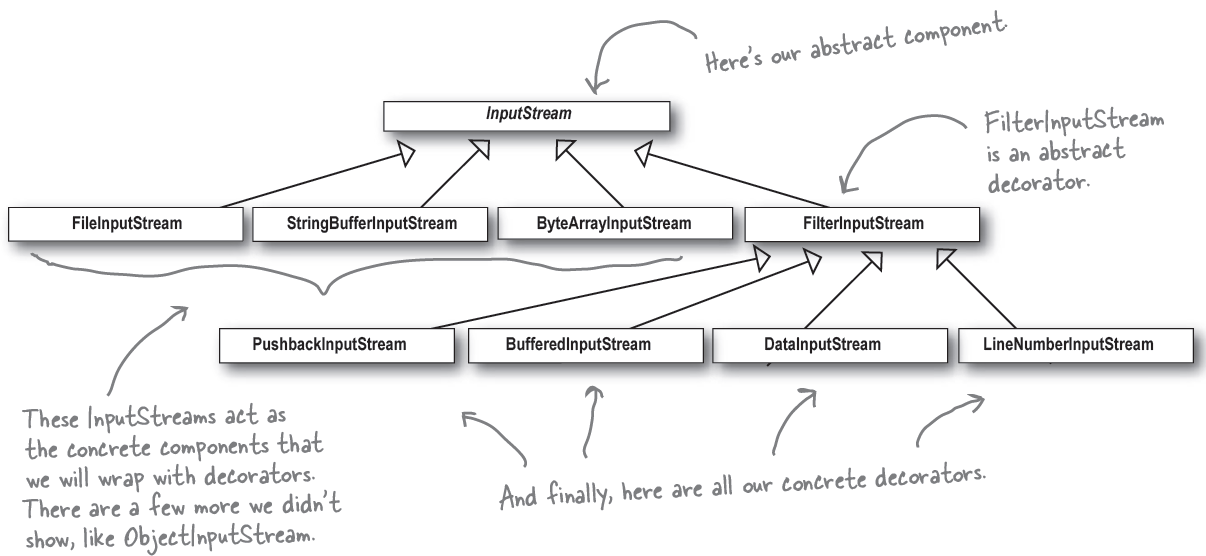
1. **Class Diagram:**



1. In decorator pattern, we extend the component class for type machining. We’re not using inheritance to get behavior. The behavior comes in through the composition of decorators with base components as well as other decorators.
2. Decorators are used with factory and builders so the creation of original components with decorators is “well-encapsulated”.
3. Java implementations:
   1. Java.io package is largely based on decorators.

**new** LineNumberInputStream(**new** BufferedInputStream(System.***in***));





* 1. As java.io demonstrates using decorator pattern may result in large number of small classes. Thus, using a decorator based API may become difficult to grasp.