The Decorator Pattern attaches additional responsibilities to an object dynamically. Decorators provide a flexible alternative to subclassing for extending functionality

***About decorators:***

* Decorators have the same supertype as the objects they decorate
* You can use one or more decorators to wrap an object.
* Given that the decorator has the same supertype as the object it decorates, we can pass around a decorated object in place of the original(wrapped) object.
* The decorator adds its own behavior either before and/or after delegating to the object it decorates to do the rest of the job.
* Objects can be decorated at any time, so we can decorate objects dynamically at runtime with as many decorators as we like.
* Decorators are typically transparent to the client of the component; that is unless the client is relying on the component’s concrete type.
* Decorators can result in many small objects in our design, and overuse can be complex.

Note: instead of abstract class, we can use interface as well

***Real world decorators – Java I/O***

InputStream

­­StringBufferInputStream

FilterInputStream

LineNumberInputStream

BufferedInputStream

PushbackInputStream

ByteArrayInputStream

DataInputStream

FileInputStream