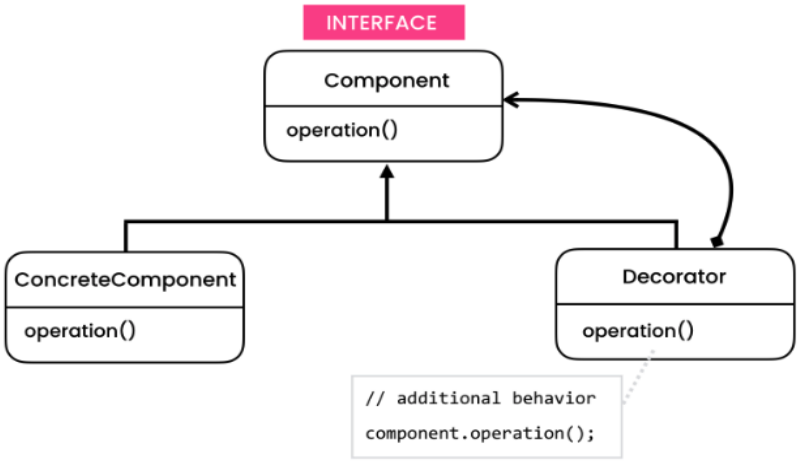
Decorator Pattern:

Definition: Adds additional behavior to an object dynamically.



Scenario: We want to implement a class that stores the data in the cloud. Later, there comes a requirement to encrypt the data before write it to the cloud. Another requirement is, sometimes files need to be compressed before writing them on the cloud.

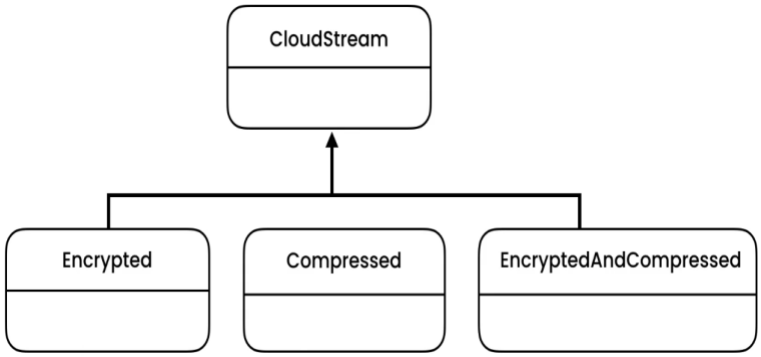
Implementation: Our cloudStream class will have the write(String data) method. Encryption class will look like

```
public class EncryptedCloudStream extends CloudStream{
   @Override
   public void write(String data) {
      var encryptedData = encrypt(data);
      super.write(encryptedData);
   }

   private String encrypt(String data){
      return "/§&§&!)§/$"; // For simplicity, imagine this is an encrypted algorithm
   }
}
```

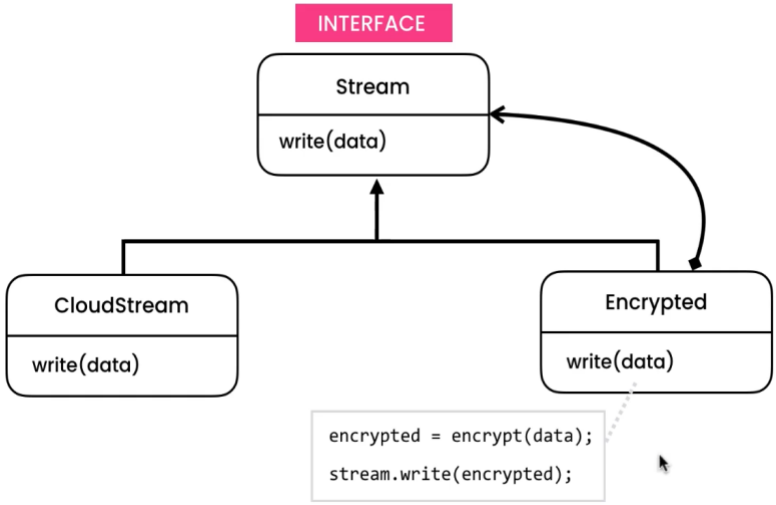
Compression class will be implemented in the same way.

Problem:



1. We have to add a new class every time for an additional manipulation on data.
2. We cannot perform combine operations efficiently, for example compression and encryption at the same time. To perform them, we will need a new combiner class.

Solution: We need to prefer composition over inheritance. Instead of using write method from base class, we can solve this by using a stream interface.




```
public interface Stream {
    void write(String data);
}
```

Use stream.write(data) instead of base.write(data) in concrete classes (Replace inheritance with composition)

```
public class EncryptedCloudStream implements Stream{
    private Stream stream; // Composition

    public EncryptedCloudStream(Stream stream) {
        this.stream = stream;
    }
```

```
    @Override
    public void write(String data) {
        var encryptedData = encrypt(data);
        stream.write(encryptedData);
    }

    private String encrypt(String data){
        return "/§&§&!)§/$"; // For simplicity, imagine this is encrypted algorithm
    }
}
```

We can make a cloud stream object and decorate it with any stream like

```
public static void storeCreditCard(Stream stream){
    stream.write("1234-1234-1234-134");
}

storeCreditCard(new EncryptedCloudStream(new CompressedCloudStream(new CloudStream())));
```