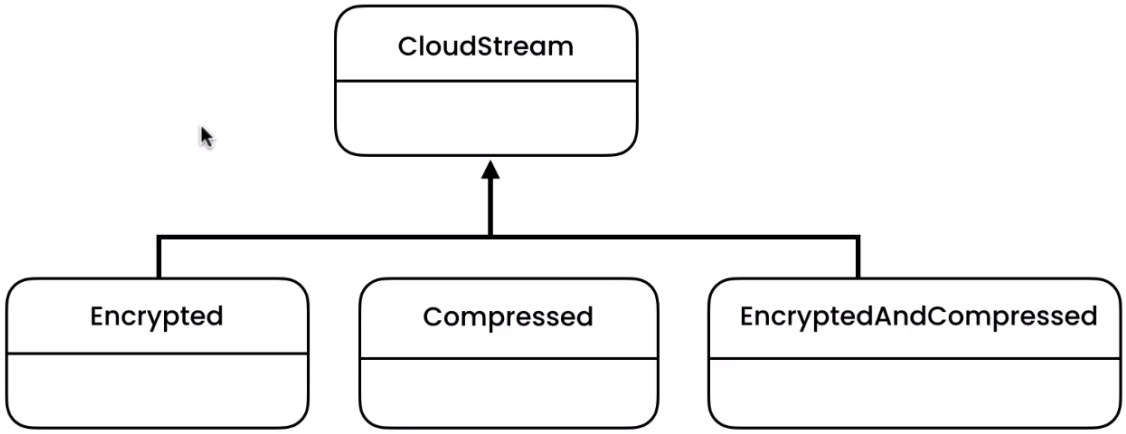
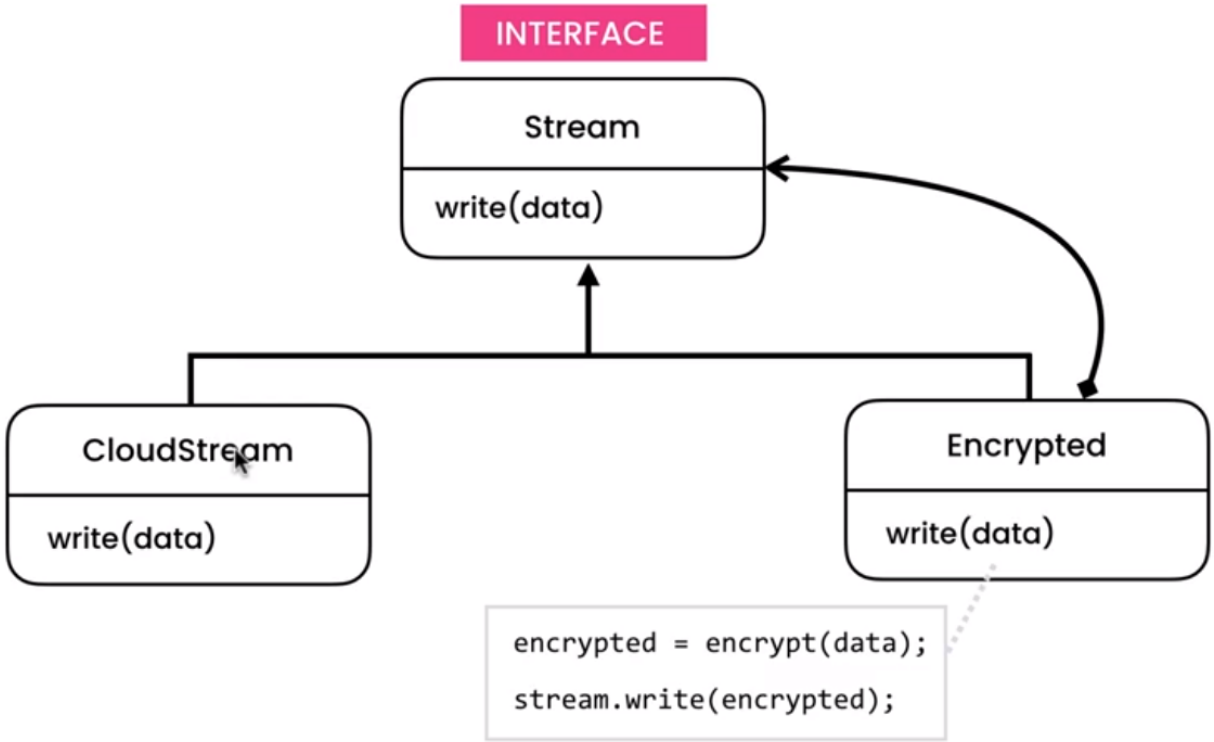
Decorator Pattern

Example Problem - You wish to add extra features / decorate an object without modifying it. You are working for a company that provides Cloud Hosting services and are asked to write a class that stores data in the class, you write and the next day are told that in some instances the data will need to be encrypted before it is stored. So you write another class that extends yesterday's class and implementthe encrytion method as well, the same thin happens the next day when you are told some data needs to be compressed before being stored. the issue comes when you are told that some data need to be encrypted and compressed before being stored in the database and we have to write another class which would carry on for each new feature.

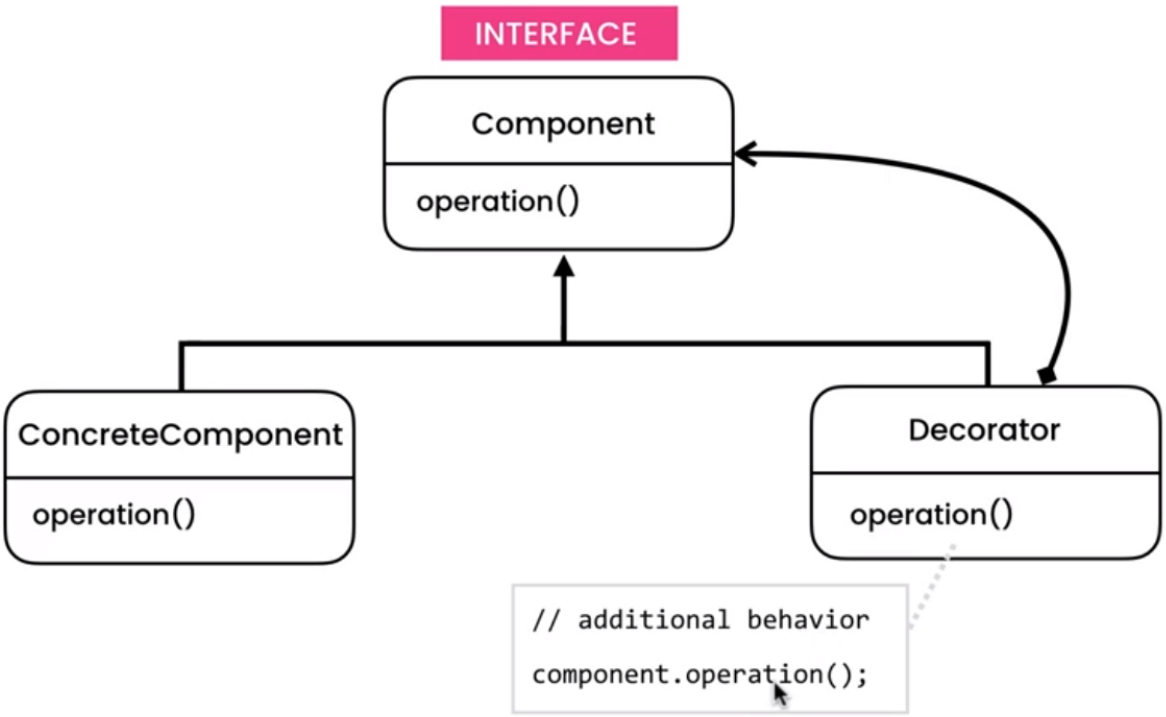
To fix this we can "decorate" the orginal CloudStrem class with feature classes. Below is the structure when inheritance is used to suppot new fdata storage features :



Using the favor composition over inheritnace approach we can implement the following :



We no longer need to call the stream method of the parent class, but only need to pass and instance of a clas which implments the stream interface. The Encrypte or Compresed clases can be used to Decorate the CloudStream component and add additional functionality as required. As per the GOF book :




```
public class CloudStream implements Stream {
  public void write(String data) {
    System.out.println("Storing " + data);
  }
}
```

```
public class CompressedCloudStream implements Stream {
  private Stream stream;

 public CompressedCloudStream(Stream stream) {
    this.stream = stream;
 }

  @Override
  public void write(String data) {
    var compressed = compress(data);
 stream.write(compressed);
 }

  private String compress(String data) {
    return data.substring(0, 5);
 }
}
```

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

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

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

  private String encrypt(String data) {
    return "!@#$(!@#*()*)(*!@#";
 }
}
```

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

```
public class Main {

public static void main(String[] args) {

//Outputs Storing 1234-1234-1234-1234

   storeCreditCard(new CloudStream());


//Outputs Storing !@#$(!@#*()*)(*!@#

   storeCreditCard(new EncyptedCloudStream(new CloudStream());


//Outputs Storing !@#$(

   storeCreditCard(new CompressedCloudStream(new EncyptedCloudStream(new CloudStream());

 }


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

}
```