Coding Dojo

“Decorator” Pattern

HSTD R&D
Satoshi Nihonyanagi

2013/8/19

課題

- 手元にあるオブジェクトには基本的な機能が用意されているが、そのオブジェクトの基本機能を実行する前や後に、何らかの処理を行いたい
  - **暗号化・復号化**
    - ファイルに書き込む機能は既にある
    - 暗号化した内容をファイルに書き込みたい
  - **圧縮・展開**
    - ファイルを読み込む機能は既にある
    - 圧縮した内容が書き込まれているファイルを展開した内容として読み込みたい。

「関数を追加する」「サブクラス化する」方法

- 圧縮してファイルに書き込む関数を追加

  →書き込むデータを圧縮する

  →書き込みメソッドを使って書き込む

- 暗号化して、ファイルに書き込む関数を追加

  →書き込むデータを暗号化

  →書き込みメソッドを使って書き込む

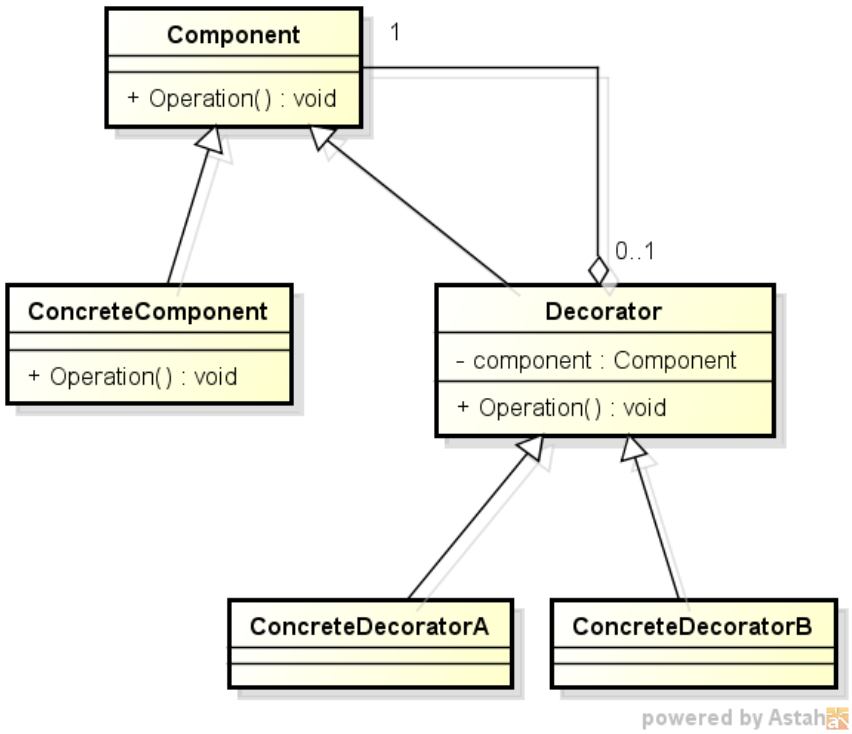
？暗号化して、圧縮したデータをファイルに書き込みたい場合は？

？メモリ上など、ファイル以外の場所に書き込みたい場合は？

→ 組み合わせによる爆発が発生

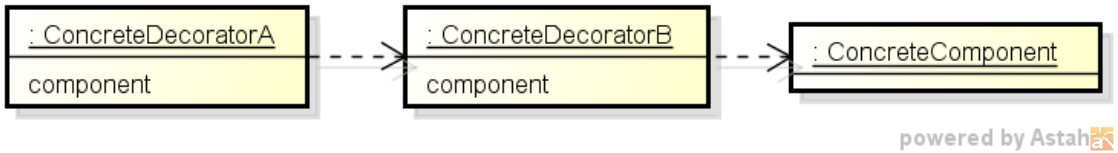
Decorator パターンによる解決策

- オブジェクトに付加的な責務を動的に付加する
- デコレータは、サブクラス化の代替となる、柔軟な機能拡張手段を提供する



サブクラスを作成せずに（継承を使用せずに）、オブジェクトに機能を追加する

コンポジションと委譲による機能拡張

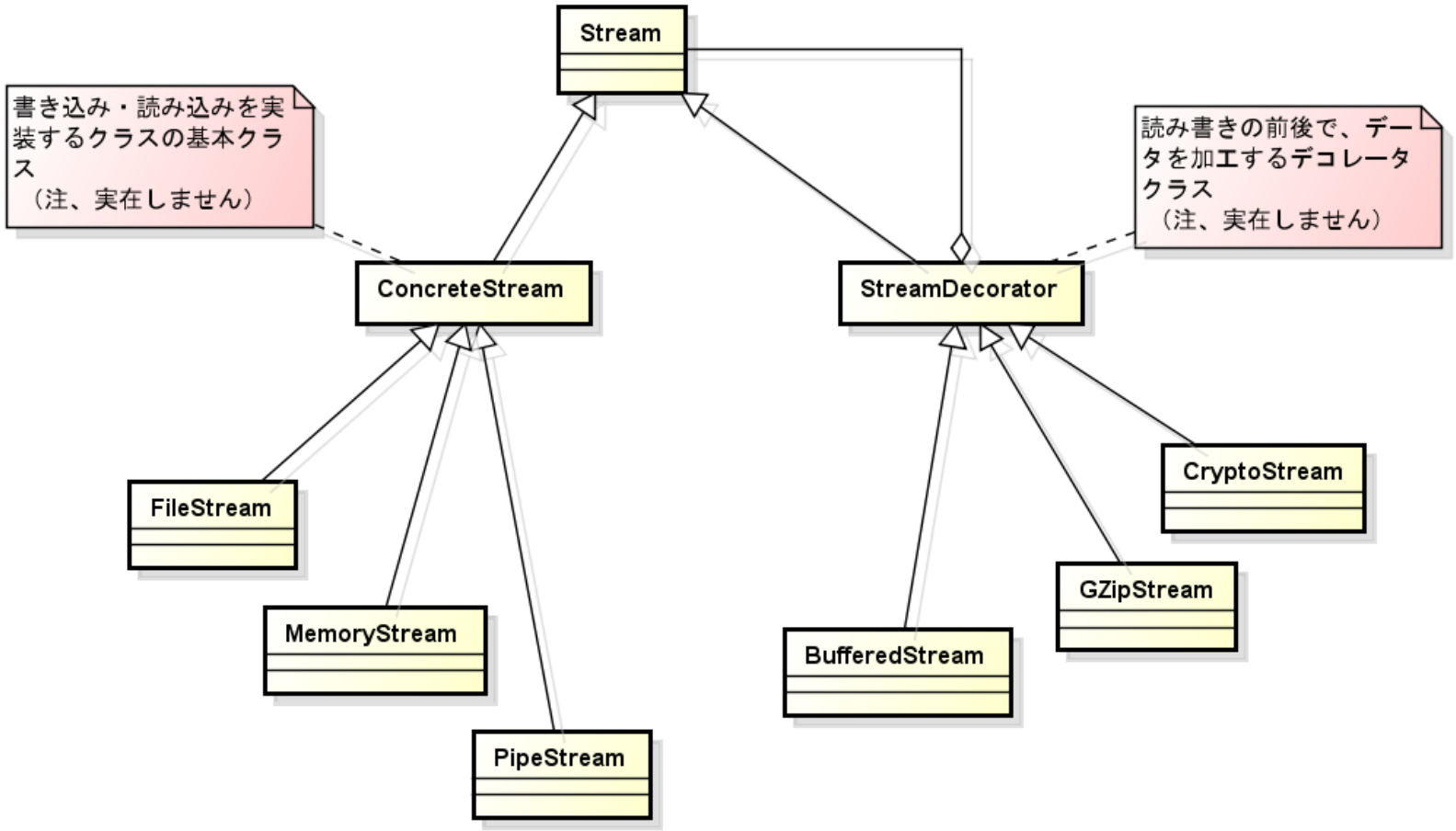


.NET Framework System.IO.Stream

ファイルを暗号化して、書き込み
➔ new CryptoStream(new FileStream(“hoge.txt”))

ファイルを暗号化して、圧縮して、書き込み
➔ new CryptoStream(new GZipStream(new FileStream(“hoge.txt”)))

データの書き込み・読み込みの前後に、データを加工する（暗号化・圧縮など）機能は、Decorator で実装されている



Decorator パターンの特徴

- デコレータは、オブジェクトをラップ（装飾）する
  → 継承ではなく、コンポジション(合成)による機能拡張
- デコレータは、ラップされたオブジェクト同じ基底クラスを持つ
  → デコレータによりラップされたオブジェクトを、ラップされたオブジェクトの代替として使用できる（利用者側からみたインターフェースは変更されない）
  → 具体的な具象クラスに依存するコードはNG

- デコレータは、ラップしたオブジェクトに処理を委譲する前後に独自の処理を追加する
- 複数のデコレータで、オブジェクトをラップできる
  → 機能拡張は、デコレータの追加、および、デコレータの組み合わせ（委譲の連鎖）により実現。既存コードは、変更不要。
  → 実行時に動的にデコレータを組み合わせて、所望の機能を合成することが可能