◇目的

再利用可能なインスタンスを保持しておくことで処理の軽量化を図る。

◇効果

処理の軽量化

◇背景

FlyweightFactory では状況によって異なる処理が行われるが、利用者側が得る結果は全く同じであるため、利用者は FlyweightFactory の内部構造を意識せずに使うことが出来る

◇Flyweightパターンの実際のコードと考え方

利用者は Flyweight クラスにあたるインスタンスを取得する場合に、直接そのクラスのコンストラクタを呼び出す代わりに FlyweightFactory() にアクセスする。一方、呼び出されたFlyweightFactory オブジェクトは、状況に応じて以下のように動作する。

その時点で対象のインスタンスが生成されていない場合

1.対象のインスタンスを新たに生成する。

2.生成したインスタンスをプールする。

3.生成されたインスタンスを返す。

対象のインスタンスが既に生成されていた場合

1.対象のインスタンスをプールから呼び出す。

2.対象のインスタンスを返す。

```
class BigChar
  def initialize(charname)
    path =
"big#{charname}.txt"
    @fontdata = ""

    IO.foreach(path) do |line|
      @fontdata +=
"#{line}¥n"
    end
  end

  def print
    puts @fontdata
  end
end
```

Flyweight

```
class BigCharFactory
  include Singleton

  def initialize
    @pool = {}
  end

  def get_big_char(charname)
    big_char = @pool[charname]

    if big_char.nil?
      big_char =
BigChar.new(charname)
      @pool[charname] = big_char
    end

    return big_char
  end
```

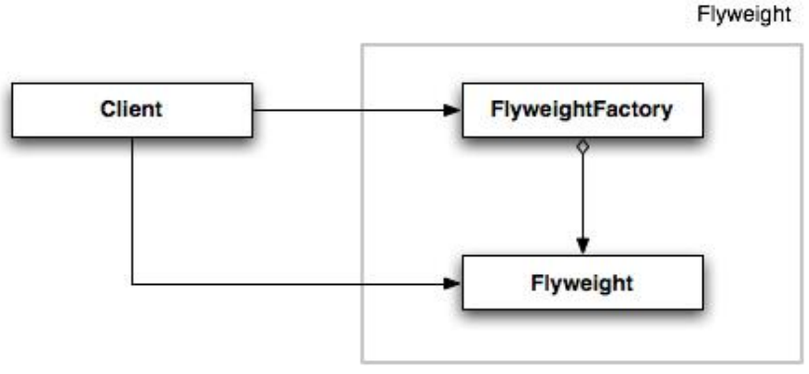
FlyweightFactory

```
class BigString
  def initialize(string)
    @bigchars = []
    factory =
BigCharFactory.instance

    string.length.times do |i|
      @bigchars <<
factory.get_big_char(string[i])
    end
  end

  def print
    @bigchars.each do |big_char|
      big_char.print
    end
  end
end
```

Client

◇問題点の改善

◇Flyweightパターンのまとめ



Clientは、FlyweightFactoryを使ってFlyweightオブジェクトを取得する。取得した後は、自由にそのオブジェクトを使える

◇注意

　一度FlyweightFactoryに保存されたインスタンスは、たとえ不要になった場合でもガベージコレクションされることがないた
よっては名示的にFlyweightFactoryから削除する必要がある。

◇総括