Flyweight Pattern
동일한 것을 공유해서 낭비를 없앤다

01. Flyweight 패턴

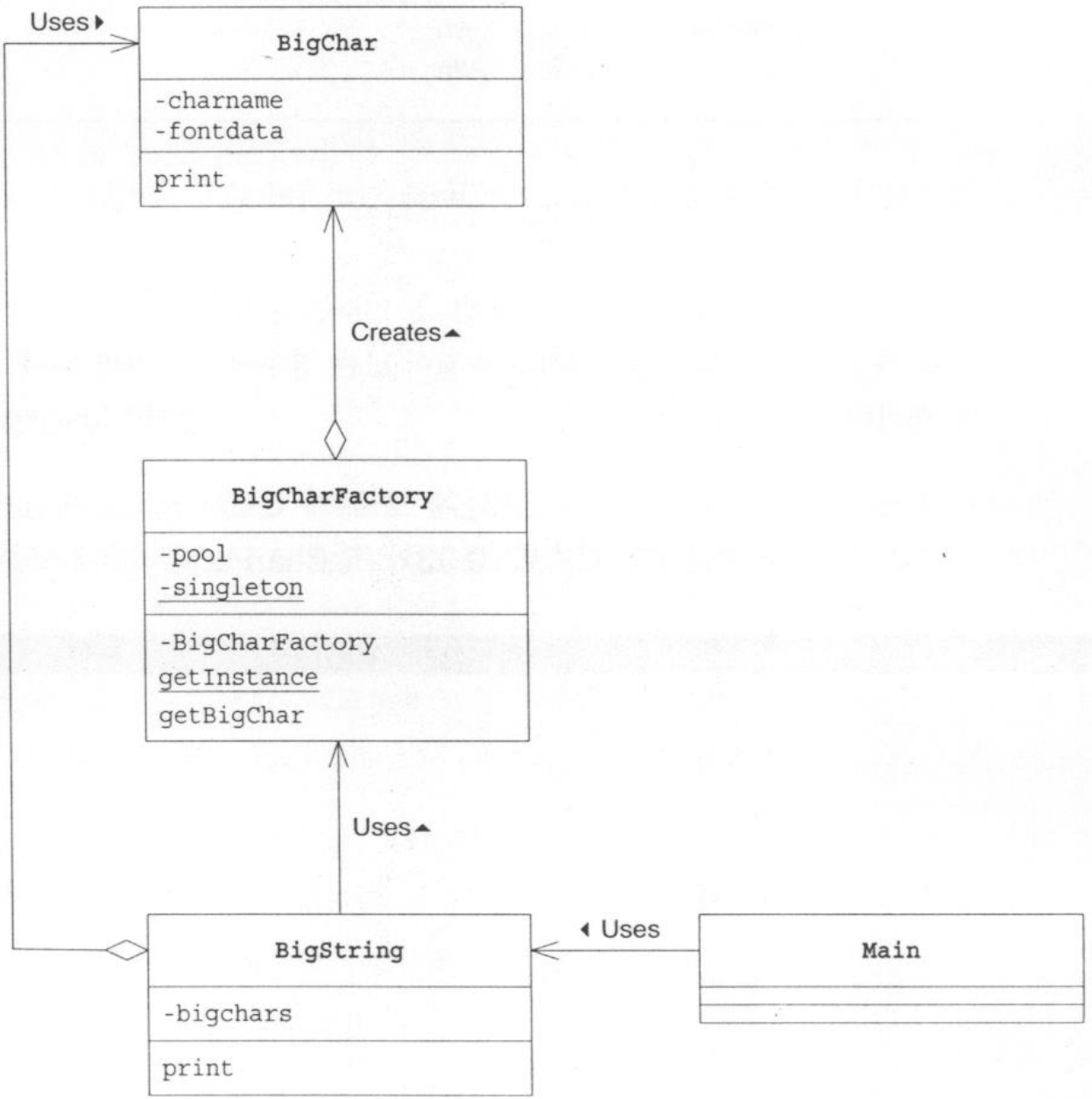
- 복싱에서의 '플라이급'을 의미한다. => 매우 가벼움
- 가볍다 == 메모리의 사용량이 적다.
- new Something( )이 실행되면, Something 클래스의 인스턴스를 보관하기 위한 공간이 주기억 장치(메인 메모리)에 할당된다.
- Flyweight 패턴의 아이디어
  - 인스턴스를 가능한 한 공유시켜서, 쓸데없이 new를 하지 않도록 한다.
  - 인스턴스가 필요할 때 마다 new를 하는 것이 아니라, 이미 만들어져 있는 인스턴스를 이용할 수 있으면 공유하자.

02. 예제 프로그램

- '큰 문자'를 출력하는 프로그램

이름	해설
BigChar	'큰 문자'를 나타내는 클래스
BigCharFactory	BigChar의 인스턴스를 공유하면서 생성하는 클래스
BigString	BigChar를 모아서 만든 '큰 문자열'을 나타내는 클래스
Main	동작 테스트용 클래스

02. 예제 프로그

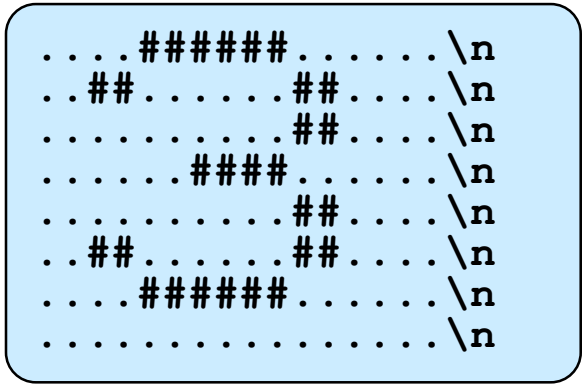

Uses
BigChar
-charname
-fontdata
print
Creates
BigCharFactory
-pool
-singleton
-BigCharFactory
getInstance
getBigChar
Uses
BigString
-bigchars
print
Uses
Main

02. 예제 프로그램

- 주요 클래스
  - BigChar
    - '큰 문자'를 나타내는 클래스
    - 파일로부터 큰 문자의 텍스트를 읽어 메모리 상에 로드하고, print 메소드에서 그것을 표시한다.
    - 큰 문자는 메모리를 많이 차지하므로, BigChar의 인스턴스를 공유하자.
  - BigCharFactory
    - BigChar 클래스의 인스턴스를 만든다.
    - 같은 문자에 대응하는 BigChar 클래스의 인스턴스가 이미 존재하는 경우에는, 이것을 이용하고 새로운 인스턴스는 만들지 않는다.
    - 지금까지 만들어진 BigChar 클래스의 인스턴스는 모두 pool이라는 필드에 보관함. (Hashtable을 이용함)

02. 예제 프로그램

- BigChar 클래스
  - '큰 문자'를 나타내는 클래스
  - 생성자
    - 인수로 제공된 문자의 '큰 문자' 버전을 작성한다.
    - 작성된 문자열은 fontdata에 저장함
    - 예를 들어, 생성자의 인수로 '3'이 주어지면, 아래 그림과 같은 문자열이 fontdata에 저장됨 <= 이 data는 big3.txt 에서 읽어온다.
      - 파일로부터 한 라인씩 읽어서, "₩n"과 함께 buf 에 추가한다.
      - buf.toString( )을 이용해서 문자열로 바꾼 다음 fontdata에 할당한다.



02. 예제 프로그램

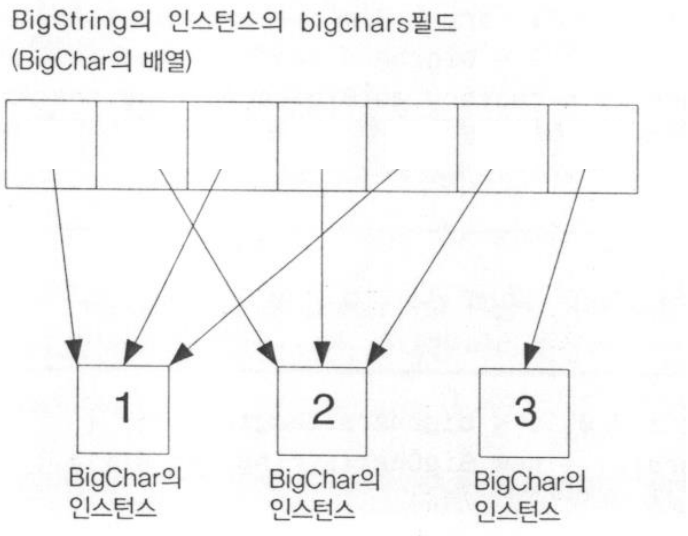
- BigCharFactory 클래스
  - BigChar의 인스턴스를 작성하는 공장
  - 여기서 인스턴스의 공유가 실현된다.
  - pool 필드:
    - 지금까지 만든 BigChar의 인스턴스들을 모아서 관리한다.
    - java.util.Hashtable을 이용함
      - "문자열(key) -> 인스턴스(value)" 대응관계를 관리해 주는 클래스
      - put(key, value)를 이용하여 문자열에 하나의 인스턴스를 대응해서 넣음.
      - get(key)를 이용하여, 문자열에 대응하는 인스턴스를 얻어옴.
  - Singleton 패턴으로 구현됨
    - 공장은 하나만 존재한다.
    - singleton 필드
    - 생성자는 private으로 선언함
    - getInstance( ) 메소드

02. 예제 프로그램

- BigCharFactory 클래스(계속)
  - getBigChar( )
    - Flyweight 패턴의 중심이 되는 메소드
    - 인수로 제공된 문자에 대응하는 BigChar 인스턴스를 만든다
      - 이미 같은 문자에 대응하는 인스턴스가 존재하면 새로운 인스턴스를 만들지 않고, 이전에 만든 인스턴스를 반환한다.

02. 예제 프로그램

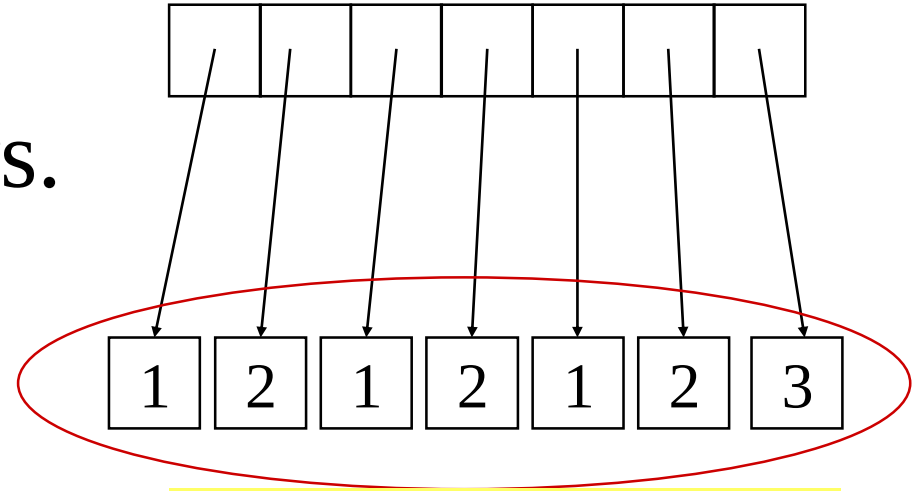
- BigString 클래스
  - BigChar들을 모은 '큰 문자열' 클래스
  - 예: "1212123"이라는 문자열에 대응하는 BigString 인스턴스



인스턴스를 공유한다.

better!

.vs.



인스턴스가 너무 많이 생긴다

worse!

02. 예제 프로그램

- BigString 클래스(계속)
  - 필드 bigchars 배열의 각 원소는 BigCharFactory로부터 BigChar를 얻어서 참조한다.
    - BigCharFactory은, 이미 생성된 BigChar인 경우에는 기존의 BigChar 인스턴스를 반환하므로, bigchars 배열의 각 원소는 같은 문자에 대해서는 BigChar 객체를 공유한다.

```
BigCharFactory factory = BigCharFactory.getInstance();
for (int i = 0; i < bigchars.length; i++) {
   bigchars[i] = factory.getBigChar(string.charAt(i));
}
```

문자열 중에서 i 번째 문자

    - (참고) 매번 새로운 인스턴스를 생성하는 경우

```
for (int i = 0; i < bigchars.length; i++) {
    bigchars[i] = new BigChar(string.charAt(i));
 }
```

02. 예제 프로그램

- Main 클래스
  - 명령행 인자로 들어온 문자열을 실인자로 해서, BigString( ) 객체를 생성한다.

03. 등장 역할

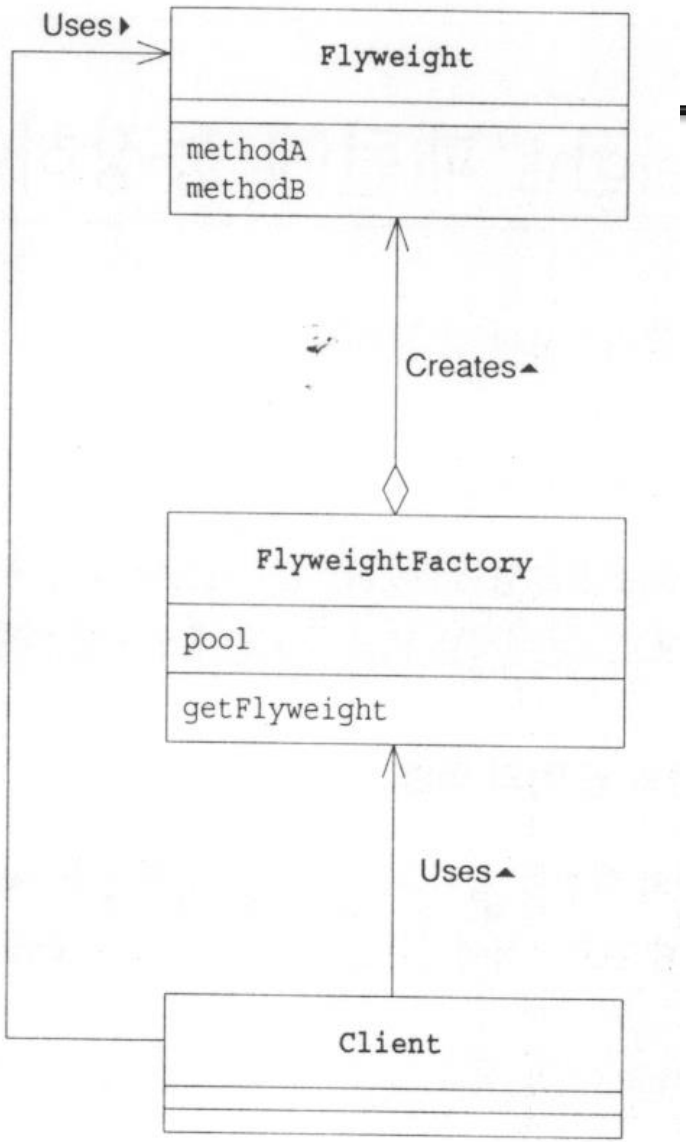
- Flyweight(플라이급)의 역할
  - 평소대로 취급하면 프로그램이 무거워져서(메모리 많이 차지해서) 공유하는게 훨씬 좋은 역할
  - 예제에서는, BigChar 클래스가 해당됨

- FlyweightFactory(플라이급의 공장)의 역할
  - Flyweight 역할을 만드는 공장의 역할
  - 공장을 사용해서 Flyweight 역할을 만들면, 인스턴스가 공유된다.
  - 예제에서는, BigCharFactory 클래스가 해당됨

03. 등장 역할

- Client(의뢰자)의 역할
  - FlyweightFactory를 사용하여 Flyweight 역할을 만들어내서 이용하는 역할
  - 예제에서는, BigString 클래스가 해당됨

03. 등장 역할


Uses
Flyweight
methodA
methodB
Creates
FlyweightFactory
pool
getFlyweight
Uses
Client

04. 독자의 사고를 넓혀주는 힌트

- 여러 곳에 영향이 미친다.
  - Flyweight 패턴의 핵심은, 인스턴스를 '공유'하는 것이다.
  - 주의해야 할 점
    - 공유하고 있는 것을 변경하면 여러 곳에 동시에 영향이 미친다.
    - => 꼭 공유시켜야 할 정보들만 Flyweight 역할에 맡겨라.
    - 예: 예제 프로그램에, '채색된 큰 문자열'을 사용하고자 할 때 색깔 정보를 어느 클래스에 두어야 할까?
      - 방법1: BigChar에 두는 방법
        - BigString 내의 공유되는 모든 BigChar 인스턴스는 동일한 색을 가지게 된다.
      - 방법2: BigString에 두는 방법
        - "몇 번째 문자는 무슨 색깔"이라는 정보를 BigString이 따로 가지고 있다.
        - 따라서, 공유되는 BigChar 인스턴스라도 다른 색깔로 표현될 수 있다.
      - 항상 어떤 정보를 공유시킬지 신중히 생각해야 한다.

04. 독자의 사고를 넓혀주는 힌트

- instrinsic(본질적인, 내재하는)과 extrinsic(부대적인, 비본질적인)
  - instrinsic한 정보: 공유되어야 하는 정보
    - 상태에 의존하지 않는 정보
    - 예: BigChar의 폰트 데이터
  - extrinsic한 정보: 공유시키지 말아야 할 정보
    - 상태에 따라 바뀌는 정보를 의미함
    - BigChar의 인스턴스가 BigString의 몇 번째 문자인가라는 정보는 BigChar가 위치하는 장소에 따라 변하기 때문에, BigChar에게 부여할 수 없다.

intrinsic한 정보	장소나 상황에 의존하지 않기 때문에 공유할 수 있습니다
extrinsic한 정보	장소나 상황에 의존하기 때문에 공유할 수 없습니다

04. 독자의 사고를 넓혀주는 힌트

- 관리되고 있는 인스턴스는 쓰레기수집(Garbage collection)이 되지 않는다.
  - BigChar 인스턴스는 Hashtable에 의해서 관리된다. 따라서, 절대 쓰레기 수집이 되지 않는다.
  - 쓰레기 수집
    - 자바는 new 연산자에 의해 객체를 생성하고, 메모리를 확보한다.
    - 객체가 많아지면, 메모리가 부족해지고, 자바 가상 기계는 쓰레기 수집이라는 처리를 시작한다.
      - 메모리 공간을 조사하며, 사용되지 않는 인스턴스를 제거하고 메모리의 빈 영역을 늘인다.

04. 독자의 사고를 넓혀주는 힌트

- 메모리 이외의 리소스
  - 프로그램에서의 리소스
    - 메모리 공간
    - 시간
      - Flyweight 패턴을 사용하면, new 횟수를 줄이므로 프로그램의 속도를 높일 수 있다.

05. 관련 패턴

- Proxy 패턴
- Composite 패턴
- Singleton 패턴

06. 요약

- 메모리의 소비를 적게 하기 위해, 인스턴스를 공유시키는 Flyweight 패턴