

5주차 퀴즈

📅 Date	@2023년 1월 5일
☰ Tags	취준스터디
🔗 출처	
📎 참고자료	

알고리즘 - 정렬

1. 퀵정렬 알고리즘을 수도코드로 작성해주세요.

```
def 퀵정렬(배열):
    배열의 길이가 1보다 작거나 같으면 :
        배열을 리턴한다
    pivot의 길이는 배열의 중앙값으로 설정한다
    pivot보다 작은값을 가지는 배열, pivot과 같은 값을 가지는 배열, pivot보다 큰 값을 가지는 배열을 초기화 한다.

    반복문, 배열에 있는 원소를 순회한다:
        만약 pivot보다 원소의 크기가 작다면 :
            pivot보다 작은값을 가지는 배열에 해당 원소를 추가합니다
        만약 pivot보다 원소의 크기가 크다면 :
            pivot보다 큰값을 가지는 배열에 해당 원소를 추가합니다
        만약 pivot과 원소의 값이 같다면 :
            pivot과 같은 값을 가지는 배열에 해당 원소를 추가합니다

    퀵정렬함수(lesser배열) + equal + 퀵정렬함수(greater배열) 을 리턴
```

2. 병합정렬 알고리즘을 수도코드로 작성해주세요.

```
def 병합정렬(배열):
    만약 배열의 길이가 2보다 작으면:
        배열을 리턴한다
    중간원소를 저장한다
    병합정렬(첫번째부터 중간까지)
    병합정렬(중간부터 마지막까지)

    앞쪽배열의 인덱스 초기화 = 0
    뒤쪽배열의 인덱스 초기화 = 0

    반복문 앞쪽그룹의 길이가 0과 같아지면 중단 그리고 뒤쪽그룹의 길이가 0과 같아지면 중단
        만약 앞쪽그룹의 원소가 뒤쪽그룹의 원소보다 작다면 :
            결과배열에 앞쪽배열의 원소를 추가한다
            앞쪽배열의 인덱스를 1추가한다
        만약 앞쪽그룹의 원소가 뒤쪽그룹의 원소보다 크다면 :
            결과배열에 뒤쪽배열의 원소를 추가한다
            뒤쪽배열의 인덱스를 1추가한다
```

결과배열에 남아있는 앞쪽배열의 원소를 이어붙인다
결과배열에 남아있는 뒤쪽배열의 원소를 이어붙인다

- 병합정렬이 어떠한 방식으로 작동하는지 설명하고 동작방식과 함께 시간복잡도가 $n \log n$ 인 이유 또한 같이 서술해주세요.

병합정렬은 현재 가지고 있는 원소를 잘게 쪼개어 그것을 조금씩 병합해 나가는 과정에서 정렬을 하는 정렬입니다.

이러한 정렬방식은 정렬한 원소의 갯수가 반씩 줄어든다는 장점을 가지고 있습니다.

전체 원소가 8개일경우 $8 \rightarrow 4 \rightarrow 2 \rightarrow 1$ 로 줄어드는데 3번의 연산이 이루어집니다. 이는 3을 $\log 8$ 과 동일하며 이 연산이 8개의 원소만큼 이루어지기 때문에 $8 \log 8$ 이라고 볼 수 있습니다. 이러한 결과를 통해 $n \log n$ 이라는 시간 복잡도를 가진다고 볼 수 있습니다.

RDBMS

- NoSQL을 특징과 함께 설명하세요

Not only SQL이라는 말의 줄임말로 비관계형 데이터베이스를 나타낸다. 기존의 관계형 데이터베이스와 달리 다른 형태의 데이터 저장 기술을 가지고 있다. 이를테면 테이블간의 관계를 정의하지 않는다는 등의 차이가 있다. 이러한 점에서 스케일 아웃(동일한 작업을 하는 테이블을 늘린다) 등의 확장에 용이하다는 장점을 가지고 있다.

또다른 NoSQL의 특징으로는 유연한 데이터를 구지면서, 새로운 필드 추가가 자유롭다는 것이 있다. 그러나 데이터 중복이 발생하는 경우가 있을 수 있으며 중복 데이터가 많기때문에 모든 컬렉션에서 수정이 필요하다. 명확한 데이터 구조를 보장하지 않는다는 단점이 있다.

장점

유연하고 자유로운 데이터 구조

새로운 필드 추가 자유로움

수평적 확장 용이

단점

데이터 중복 발생가능

중복 데이터가 많기 때문에 데이터 변경시 모든 컬렉션에서 수정 필요하다

명확한 데이터 구조를 보장하지 않음

- RDB와 NoSQL은 각각 어떤 서비스나 기능에 적합할지 예를 들고 이유를 설명하세요

알아보기 쉽지만 제한적인 형태 vs 알아보기 어렵지만 유연한 형태

관계형데이터베이스는 데이터 구조가 변경될 여지가 적고 명확한 경우에 적합하다. 일반적인 웹서비스 데이터베이스 구조를 설계해놓고 운영하기 때문에 적합하다

반면에 비관계형데이터베이스의 경우 정확한 데이터 구조를 알기 어렵고 데이터가 변경 혹은 확장이 될 수 있는 경우 적합하다. 많은 데이터를 처리해야하는 서비스 혹은 유저의 행위를 기록해야하는 경우 혹은 처리과정을 진행하기 전 데이터를 저장하기에 적합하다. Redis같은 경우 좋아요 시스템에 많이 사용된다.

디자인 패턴

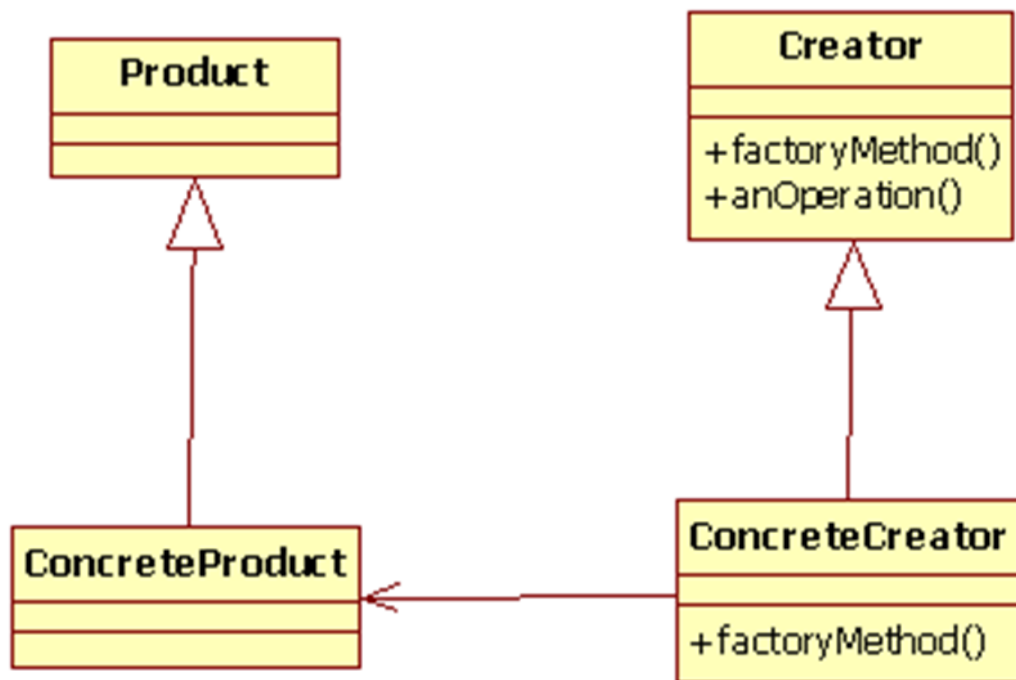
1. 싱글톤 패턴에 대해 설명하고 DBCP(database connection pool)에서 어떤 방식으로 적용이 되는 지 간단하게 설명하세요

커넥션 풀에서 자원을 사용한다는 스레드에게 자원을 건네줍니다. 즉, 스레드가 커넥션 풀에 요청을 하면 풀에서 자원을 할당해주는 것입니다. 이는 커넥션 자원을 각각의 스레드가 접근해서 사용하는 방식과 달리 커넥션 자원을 관리하는 커넥션 풀이라는 중간자가 있기에 가능한 것입니다.

디비 커넥션 같은 경우 속도가 걸리는 작업, 부하가 걸리는 작업이기 때문에 미리 커넥션을 잡아놓고 그 풀을 사용하는 서비스한테만 할당을 해줘서 사용하기 위해서 싱글톤 → 메모리를 공유하고 미리 메모리를 할당해놓을 수 있기 때문에 커넥션을 한곳에서 관리할 수 있다는 장점이 있다.

실제로 생성된 객체는 하나임 !! → DBCP 같은 경우 커넥션 하나를 어떤 하나가 관리를 해서 시간이 오래걸리는 커넥션 작업의 효율성을 높임

2. 다음은 디자인 패턴 중 하나의 기본 구조도 입니다. 어떤 패턴의 구조도인지 설명하고 작동원리를 설명하세요



팩토리 메서드 패턴

객체를 생성하기 위한 인터페이스를 정의하는데 있어서 어떤 클래스의 인스턴스를 만들지는 서브클래스에서 결정하게 만드는 것이다. 즉, 클래스의 인스턴스를 만드는 일을 서브클래스에게 맡기는 것을 말한다.

팩토리 메서드 → 의존성 주입과 관련이 있다. 구상 클래스에 대한 불필요한 의존성을 줄여줌으로써 느슨한 결합을 지향하는 기법이다.

이는 SOLID(DIP)의 의존관계역전 원칙에 기인한 패턴이다.

3. PPT에서 발송클래스의 메소드인 택배사에게발송행위를 전략패턴으로 재구성하였습니다.
해당 전략(택배사에게발송)을 동적으로 지정할 수 있도록 발송 클래스를 수정해주세요

사진 1. 행위 인터페이스 구성

```

interface 택배사에게발송행위 {

    택배사에게발송 () : boolean
}

class 일본으로발송하기 implements 택배사에게발송행위 {
    택배사에게발송 = () => {
        //일본택배사에게 배송
    }
}

```

사진 2. 발송 클래스

```

class 발송 {
    발송상품
    택배사에게발송행 : 택배사에게발송행위

    발송(상품) {
        발송상품 = 상품
    }

    택배사에게발송 = () => {
        택배사에게발송행.택배사에게발송()
    }
}

```

발송국가 = 국가코드

국가코드에 따른 분기문 작성 ?

사진 3. 발송 클래스를 상속하여 사용하는 로직

```

class 국내발송 extends 발송 {

  발송(상품) {
    발송상품 = 상품
    택배사에게발송행 = new 국내택배사에게발송하기()
  }

  택배사에게발송 = () => {
    택배사에게발송행.택배사에게발송()
  }
}

class 일본발송 extends 발송 {

  발송(상품) {
    발송상품 = 상품
    택배사에게발송행 = new 일본으로발송하기()
  }

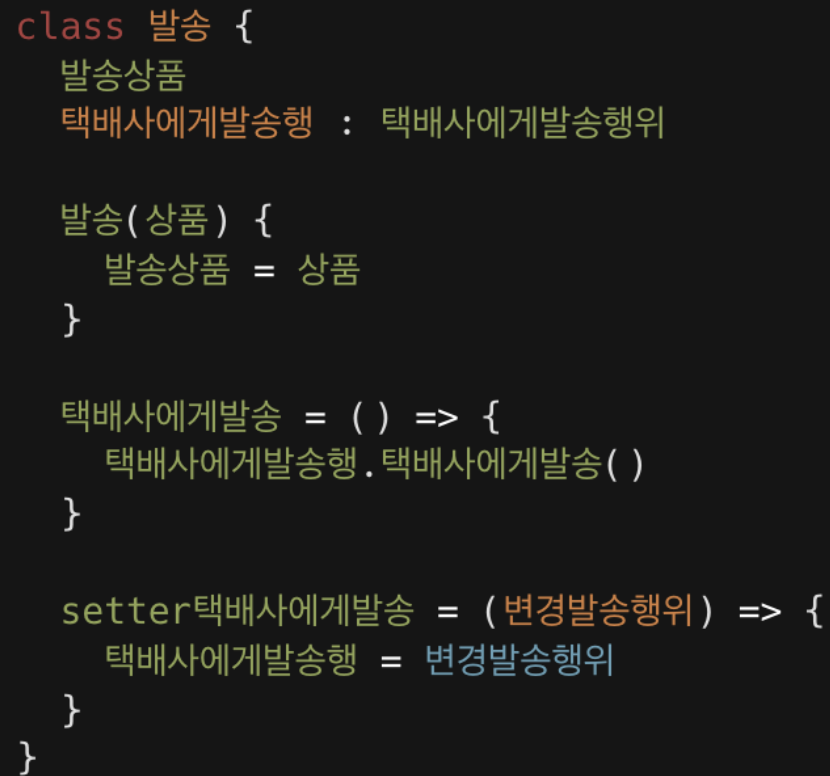
  택배사에게발송 = () => {
    택배사에게발송행.택배사에게발송()
  }
}

=====
let 발송Obj
if(상품.nationCode == 'KR') {
  발송Obj = new 국내발송()
}else {
  발송Obj = new 일본발송()
}

-----프로세스 -----

발송Obj.택배사에게로()

```



```

class 발송 {
    발송상품
    택배사에게발송행 : 택배사에게발송행위

    발송(상품) {
        발송상품 = 상품
    }

    택배사에게발송 = ( ) => {
        택배사에게발송행.택배사에게발송( )
    }

    setter택배사에게발송 = ( 변경발송행위 ) => {
        택배사에게발송행 = 변경발송행위
    }
}

```

- 자주 변하는 부분을 인터페이스를 구현해놓고 구체화해서 구현을 함
- 동적으로 사용할 가능성이 높은 경우 사용한다.