REPORT

LAB 번호 : HW2

과목 및 분반 : 자바프로그래밍2 2분반

제출일 : 2025.09.23

학번 : 32203919 (컴퓨터공학과)

이름 : 장천명

DKU 단국대학교 DANKOOK UNIVERSITY

1. 프로젝트의 목적

Strategy pattern을 활용해서 다양한 텍스트 정규화 및 토큰화 방법을 조합할 수 있는 유연한 텍스트 처리 시스템을 구현

---

2. 시스템 흐름 및 데이터 흐름

1. [App.java]

- 테스트 텍스트 입력
- 두 개의 파이프라인 설정 (정규화기 리스트, 토큰화기, 저장소)

2. [CountableWordProcessor.java]

- 텍스트 처리 파이프라인 시작
- 정규화 → 토큰화 → 저장 과정 조율

3. [정규화 단계] Strategy 패턴으로 순차 적용

- 파이프라인 1: NFKCTextNormalizer → LowercaseTrimTextNormalizer
- 파이프라인 2: AsciiFoldingNormalizer → SimpleStemNormalizer

4. [토큰화 단계] Strategy 패턴으로 텍스트 분리

- 파이프라인 1: RegexTokenizer (공백+구두점 기준)
- 파이프라인 2: WhitespaceTokenizer (공백만 기준)

5. [데이터 모델 생성]

- [Word.java] 각 토큰을 Word 객체로 생성
- [CountableWord.java] Word와 빈도 정보를 함께 관리

6. [CountableWordRepository.java]

- 초성별 그룹화 (첫 글자를 키로 사용)
- 빈도 계산 및 관리 (중복 단어 처리)

7. [ConsolePrinter.java]

- TreeMap으로 키(초성) 정렬
- Stream API로 값(단어) 정렬
- 포맷된 결과 출력

8. [Your Code 기능]

- 전체 단어 개수 계산 (중복 포함)

---

3. 핵심 기능

App : 파이프라인 생성, 텍스트 처리, 결과 출력

CountableWordProcessor : 정규화 → 토큰화 → 저장 과정 조율

CountableWordRepository : 단어 추가/제거, 빈도 계산, 초성별 조회

ConsolePrinter : 키 정렬, 값 정렬, 포맷된 출력

---

4. 주요 코드

텍스트 처리 파이프라인 - 전체 텍스트 처리 과정을 관리하는 핵심 메서드

```java
public void processText(String text) {
    // 입력 검증
    if (text == null || text.isEmpty()) return;

    // 정규화 단계: 모든 정규화 전략을 순서대로 적용
    for (ITextNormalizer norm : textNorms) {
        text = norm.normalize(text);
    }

    // 토큰화 단계: 정규화된 텍스트를 단어들로 분리
    List<String> tokens = tokenizer.tokenize(text);

    // 저장 단계: 각 토큰을 저장소에 추가 (빈도 계산)
    for (String token : tokens) {
        if (!token.isEmpty()) {
            repository.addOne(token);
        }
    }
}
```

1. **입력 검증**: null이나 빈 문자열 체크
2. **정규화**: Strategy 패턴으로 순차적으로 정규화 적용
3. **토큰화**: 정규화된 텍스트를 단어들로 분리
4. **저장**: 각 토큰을 저장소에 추가하여 빈도 계산

Strategy 패턴으로 다양한 정규화/토큰화 방법을 유연하게 조합

초성별 그룹화 저장 - 단어를 초성별로 그룹화하여 저장하고 빈도를 계산

```
public void print(Map<Character, List<CountableWord>> grouped) {
    // 키(초성) 정렬: TreeMap 사용
    Map<Character, List<CountableWord>> sortedGrouped = new TreeMap<>(grouped);

    for (Map.Entry<Character, List<CountableWord>> entry : sortedGrouped.entrySet()) {
        System.out.print(entry.getKey() + ": ");

        // 각 리스트 내의 단어들을 알파벳순으로 정렬
        List<CountableWord> sortedWords = entry.getValue().stream()
            .sorted((cw1, cw2) -> cw1.getWord().toString().compareTo(cw2.getWord().toString()))
            .collect(Collectors.toList());

        // 정렬된 단어들을 출력
        for (CountableWord cw : sortedWords) {
            System.out.print(cw + " ");
        }
        System.out.println();
    }
}
```

1. **키 생성**: 단어의 첫 글자를 키로 사용
2. **리스트 관리**: computeIfAbsent()로 해당 키의 리스트 생성/조회
3. **중복 처리**: 같은 단어가 있으면 빈도 증가, 없으면 새로 추가

정렬된 결과 출력 - 저장된 데이터를 정렬하여 콘솔에 출력

```
public void print(Map<Character, List<CountableWord>> grouped) {
    // 키(초성) 정렬: TreeMap 사용
    Map<Character, List<CountableWord>> sortedGrouped = new TreeMap<>(grouped);

    for (Map.Entry<Character, List<CountableWord>> entry : sortedGrouped.entrySet()) {
        System.out.print(entry.getKey() + ": ");

        // 각 리스트 내의 단어들을 알파벳순으로 정렬
        List<CountableWord> sortedWords = entry.getValue().stream()
            .sorted((cw1, cw2) -> cw1.getWord().toString().compareTo(cw2.getWord().toString()))
            .collect(Collectors.toList());

        // 정렬된 단어들을 출력
        for (CountableWord cw : sortedWords) {
            System.out.print(cw + " ");
        }
        System.out.println();
    }
}
```

---

5. Your code

```java
public int getTotalWordCount() {
    int totalCount = 0;

    // 모든 초성 그룹을 순회
    for (List<CountableWord> wordList : map.values()) {
        // 각 그룹의 모든 단어를 순회
        for (CountableWord cw : wordList) {
            // 각 단어의 등장 횟수를 총합에 추가
            totalCount += cw.getCount();
        }
    }

    return totalCount;
```

CountableWordRepository에 위치한 yourcode이다.

1. 모든 초성 그룹을 순회
2. 각 그룹의 모든 단어를 순회
3. 각 단어의 등장 횟수를 총합에 추가

이렇게 중첩 반복문으로 모든 등장 횟수의 합계를 계산한다.

```java
public int getTotalWordCount() {
    return repository.getTotalWordCount();
```

CountableWordProcessor에 위치한 yourcode이다.

Repository의 기능을 외부에 노출하는 wrapper 메서드이다.

---

6. 결과 및 분석

Pipeline1 결과

```
[Pipeline #1] 초성별 빈도 (CountableWord 적용):
1: 13(1)
3: 3가지(1)
a: abc(1)
b: book(1) brown(1)
c: caf?(1) com(1)
d: docs(1) dogs(1)
e: email(1) example(2)
f: fox(1) full(1)
h: handle(1) hashtag(1) https(1)
j: jumps(1)
l: lazy(1)
n: na?ve(1) nfkc(1)
o: o(1) one(1) org(1) over(1)
q: quick(1) quick?brown?fox(1) quotes(1)
r: re(1) reading(1) reilly(1)
s: s(1)
t: test(1) three(1) two(1)
v: visit(1)
w: width(1)
y: you(1)
미: 미나(1)
프: 프로그래밍?좋아요(1)
항: 항목(1)
파이프라인 1 전체 단어 개수: 41
```

Pipeline2 결과

```
[Pipeline #2] 초성별 빈도 (CountableWord적용):
": "quotes",(1)
#: #hashtag(1)
(: (nfkc:(1)
1: 13(1)
@: @handle(1)
b: book...(1) brown(1)
c: cafe,(1)
d: dogs.(1)
e: email:(1)
f: fox!!(1) full-width(1)
h: https://example.com/docs.(1)
j: jumps(1)
l: lazy(1)
n: naive)(1)
o: o'reilly's(1) one,(1) over(1)
q: quick,(1) quick?brown?fox;(1)
r: reading(1)
t: test@example.org,(1) three.(1) two,(1)
v: visit(1)
y: you're(1)
미: 미나(1)
프: 프로그래밍?좋아요!(1)
항: 항목:(1)
３: ３가지(1)
ａ: ａｂｃ,(1)
파이프라인 2 전체 단어 개수: 32
```

결과 분석

파이프라인 1 (NFKC + 정규식 토큰화)

- 전체 단어 개수: 41개(Your Code)
- 특징: 정규식으로 공백과 구두점을 기준으로 정확한 단어 분리
- 결과: 더 많은 토큰 생성, 깔끔한 단어 분리

파이프라인 2 (ASCII 폴딩 + 공백 토큰화)

- 전체 단어 개수: 32개(Your Code)
- 특징: 공백만을 기준으로 분리하여 구두점이 포함된 토큰 생성
- 결과: 더 적은 토큰, 구두점이 포함된 형태