**REPORT**

자바프로그래밍2

Lab 4.

|  |  |
| --- | --- |
| **제출일** | 2025.10.15 |
| **소속** | 컴퓨터공학과 |
| **학번** | 32202870 |
| **이름** | 유위창 |

## 구현 파일 구조

* IPOI.java (Component 인터페이스) yourcode포함
* POI.java (Concrete Component) yourcode포함
* POIDecorator.java (Abstract Decorator) yourcode포함
* CategoryPOIDecorator.java (카테고리 데코레이터) yourcode포함
* HashtagPOIDecorator.java (해시태그 데코레이터) yourcode포함
* DistancePOIDecorator.java (거리 데코레이터) yourcode 포함
* ProximityPOIDecorator.java (근접성 데코레이터) yourcode포함
* POIGsonFileLoader.java (JSON 로더) yourcode포함
* GeoUtil.java (거리 계산 유틸리티)
* Location.java (위치 데이터 모델)
* App.java (메인 애플리케이션)

# 구현 내용

## IPOI.java(yourcode)

### 코드텍스트, 폰트, 라인, 스크린샷이(가) 표시된 사진 AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

### 설명

* 역할: Decorator 패턴의 Component 인터페이스
* 목적: 모든 POI 객체와 데코레이터가 구현해야 하는 공통 계약 정의
* 메서드: getInformation()은 POI 정보를 문자열로 반환
* 중요성: 이 인터페이스를 통해 POI와 데코레이터를 동일하게 취급 가능

## POI.java(yourcode)

### 코드텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진 AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

### 설명

* 역할: Decorator 패턴의 Concrete Component (기본 구성 요소)
* 생성자: name과 location을 받아 POI 객체 생성(yourcode)
* getInformation(): “POI: [이름] @ [위치]” 형식으로 기본 정보 반환
* Getter 메서드: JSON 파싱과 데코레이터에서 사용하기 위한 접근자 메서드
* 중요성: 이 클래스가 데코레이터로 감싸질 기본 객체 역할

## POIDecorator.java(yourcode)

### 코드텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진 AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

### 설명

* 역할: 추상 데코레이터 클래스, 모든 구상 데코레이터의 부모 클래스
* decoratedPOI 필드: 감싸고 있는 IPOI 객체를 저장 (다른 데코레이터일 수도 있고 POI일 수도 있음)
* getInformation(): 기본적으로 감싸고 있는 객체의 정보를 그대로 반환 (하위 클래스에서 오버라이드하여 정보 추가)
* unwrapPOI() 메서드:
  + 목적: 데코레이터 체인을 따라 내려가서 원본 POI 객체를 추출
  + 동작 원리: 재귀 호출을 사용하여 데코레이터를 하나씩 벗겨냄
  + 필요성: DistancePOIDecorator와 ProximityPOIDecorator에서 POI의 위치 정보가 필요하지만, IPOI 인터페이스에는 getLocation()이 없으므로 원본 POI를 찾아야 함
* 재귀 과정 예시:
* ProximityPOIDecorator  
   → DistancePOIDecorator  
   → HashtagPOIDecorator  
   → CategoryPOIDecorator  
   → POI (원본 발견!)

## CategoryPOIDecorator.java

### 코드텍스트, 폰트, 번호, 소프트웨어이(가) 표시된 사진 AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

### 설명

* 역할: POI에 카테고리 정보를 추가하는 데코레이터
* 생성자: 감쌀 IPOI 객체와 카테고리 문자열을 받음
* getInformation():
  + super.getInformation()으로 감싸고 있는 객체의 정보를 먼저 가져옴
  + 그 뒤에 ” | Category: [카테고리]” 형식으로 정보 추가
* 출력 예시: POI: 경복궁 @ 37.577, 126.977 | Category: 궁궐

## HashtagPOIDecorator.java

### 코드텍스트, 스크린샷, 폰트이(가) 표시된 사진 AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

### 설명

* 역할: POI에 해시태그 정보를 추가하는 데코레이터
* 생성자: 감쌀 IPOI 객체와 해시태그 리스트를 받음
* getInformation():
  + 해시태그가 null이거나 비어있으면 원본 정보만 반환
  + 해시태그가 있으면 Stream API의 Collectors.joining(", ")을 사용하여 쉼표로 구분된 문자열로 변환
  + ” | Hashtags: [태그1, 태그2, …]” 형식으로 정보 추가
* Stream API 활용: 리스트를 간결하게 하나의 문자열로 결합
* 출력 예시: POI: 경복궁 @ 37.577, 126.977 | Category: 궁궐 | Hashtags: [#경복궁, #궁궐, #서울]

## DistancePOIDecorator.java

### 코드텍스트, 스크린샷, 폰트, 라인이(가) 표시된 사진 AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

### 설명

* 역할: POI와 현재 위치 사이의 거리 정보를 추가하는 데코레이터
* 생성자: 감쌀 IPOI 객체와 현재 위치를 받음
* getInformation():
  + unwrapPOI() 사용: decoratedPOI는 IPOI 타입이므로 getLocation()이 없음. unwrapPOI()로 원본 POI를 추출하여 위치 정보 접근
  + 거리 계산: GeoUtil.distanceMeters()를 사용하여 Haversine 공식으로 정확한 지구 표면 거리 계산
  + 포맷팅: String.format()으로 소수점 2자리까지 표시
  + ” | Distance: [거리]m” 형식으로 정보 추가
* unwrapPOI()가 필요한 이유: 데코레이터 체인 속에서 원본 POI의 위치 정보가 필요하기 때문
* 출력 예시: ... | Distance: 321.67m

## ProximityPOIDecorator.java

### 코드텍스트, 스크린샷, 폰트이(가) 표시된 사진 AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

### 설명

* 역할: POI가 현재 위치로부터 특정 반경 내에 있는지 판단하는 데코레이터
* 생성자: 감쌀 IPOI 객체, 현재 위치, 그리고 반경(미터)을 받음
* getInformation():
  + unwrapPOI() 사용: 원본 POI를 추출하여 위치 정보 접근
  + 거리 계산: GeoUtil.distanceMeters()로 거리 계산
  + 근접성 판단: 계산된 거리가 반경보다 작거나 같으면 “Yes”, 아니면 “No”
  + 삼항 연산자: (distance <= radiusMeter) ? "Yes" : "No"로 간결하게 구현
  + ” | Within [반경]m: [Yes/No]” 형식으로 정보 추가
* 활용: 200m 반경 내의 POI만 필터링하거나, 근처에 있는 POI를 강조 표시할 때 유용
* 출력 예시: ... | Distance: 67.21m | Within 200m: Yes

## POIGsonFileLoader.java

### 코드텍스트, 폰트, 라인, 스크린샷이(가) 표시된 사진 AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

텍스트, 폰트, 스크린샷이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

### 설명

* 역할: JSON 파일에서 POI 데이터를 로드하고, 선택적으로 데코레이터 적용

#### load() 메서드

* 목적: 기본 POI 리스트만 로드
* 구현:
  + Gson 라이브러리를 사용하여 JSON 파일 파싱
  + TypeToken을 사용하여 {"pois": [...]} 구조의 JSON을 Map<String, List>로 변환
  + “pois” 키에 해당하는 POI 리스트 반환
* try-with-resources: FileReader가 자동으로 닫히도록 보장
* 예외 처리: 파일 읽기 오류 시 빈 리스트 반환

#### loadAsDecorated() 메서드

* 목적: POI를 로드하고 Category와 Hashtag 데코레이터를 자동으로 적용
* 구현 과정:
  1. JSON 파일에서 POI 리스트 로드
  2. 각 POI에 대해:
     + 기본 POI 객체 생성 (name, location만 사용)
     + Category가 있으면 CategoryPOIDecorator로 감싸기
     + Hashtag가 있으면 HashtagPOIDecorator로 감싸기
  3. 데코레이트된 IPOI 리스트 반환
* 데코레이터 체인 생성:
* POI → CategoryPOIDecorator → HashtagPOIDecorator
* null 체크: category나 hashtags가 null이거나 empty일 경우 데코레이터를 적용하지 않음
* 유연성: 일부 POI는 category만, 일부는 hashtag만, 일부는 둘 다 가질 수 있음

## GeoUtil.java

### 코드텍스트, 폰트, 스크린샷, 번호이(가) 표시된 사진 AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

### 설명

* 역할: 두 지점 간의 실제 거리를 계산하는 유틸리티 클래스
* Haversine 공식: 지구를 구로 가정하고 두 위도/경도 좌표 사이의 최단 거리(대원 거리) 계산
* 구현 단계:
  1. 위도와 경도의 차이를 라디안으로 변환
  2. Haversine 공식 적용하여 중심각 계산
  3. 지구 반지름(6371km)을 곱하여 실제 거리 계산
* 반환값: 미터 단위의 거리
* 정확도: 실제 지구 표면의 거리를 정확하게 계산 (직선 거리가 아님)
* 사용처: DistancePOIDecorator와 ProximityPOIDecorator에서 사용

## App.java (동적 데코레이터 적용)

### 코드텍스트, 폰트, 라인, 스크린샷이(가) 표시된 사진 AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

### 설명

* 역할: 실행 시간에 동적으로 데코레이터 추가
* 동작 과정:
  1. decoratedBase는 이미 Category와 Hashtag 데코레이터가 적용된 IPOI 리스트
  2. 각 POI에 대해:
     + 현재 위치(cur)를 사용하여 DistancePOIDecorator로 감싸기
     + 그 위에 ProximityPOIDecorator(반경 200m)로 감싸기
  3. 최종 데코레이터 체인을 통해 정보 출력
* 최종 데코레이터 체인:
* POI  
   → CategoryPOIDecorator  
   → HashtagPOIDecorator  
   → DistancePOIDecorator  
   → ProximityPOIDecorator
* getInformation() 호출 흐름:
* 1. ProximityPOIDecorator.getInformation() 호출  
  2. super.getInformation() → DistancePOIDecorator.getInformation()  
  3. super.getInformation() → HashtagPOIDecorator.getInformation()  
  4. super.getInformation() → CategoryPOIDecorator.getInformation()  
  5. super.getInformation() → POI.getInformation()  
  6. POI가 "POI: 경복궁 @ ..." 반환  
  7. CategoryPOIDecorator가 " | Category: 궁궐" 추가  
  8. HashtagPOIDecorator가 " | Hashtags: [...]" 추가  
  9. DistancePOIDecorator가 " | Distance: 67.21m" 추가  
  10. ProximityPOIDecorator가 " | Within 200m: Yes" 추가

## Category와 Hashtag 데코레이터 적용

== Base decorated (from JSON) ==  
 - POI: 경복궁 향원정 @ 37.5806903, 126.9769632 | Category: 정자 |  
 Hashtags: [#경복궁, #향원정, #정자, #연못, #서울, #문화재]  
 - POI: 경복궁 근정전 @ 37.5806903, 126.9769632 | Category: 정전 |  
 Hashtags: [#경복궁, #근정전, #정전, #조선, #왕궁, #문화재]  
...

설명: loadAsDecorated() 메서드로 JSON에서 로드하면서 자동으로 Category와 Hashtag 데코레이터가 적용됨

## 경로를 따라 이동하며 Distance와 Proximity 데코레이터 적용

== Along path ==  
  
-- step @ (37.57790, 126.97600)  
 -> POI: 경복궁 @ ... | Category: 궁궐 | Hashtags: [...] |  
 Distance: 105.94m | Within 200m: Yes  
 -> POI: 경복궁 근정문 및 행각 @ ... | Category: 문/행각 | Hashtags: [...] |  
 Distance: 67.21m | Within 200m: Yes  
  
-- step @ (37.57840, 126.97660)  
 -> POI: 경복궁 근정문 및 행각 @ ... | Category: 문/행각 | Hashtags: [...] |  
 Distance: 30.22m | Within 200m: Yes

설명: - 위치가 바뀌면서 동일한 POI의 거리가 동적으로 변화 (67.21m → 30.22m) - 200m 반경 내에 있는지 실시간으로 판단 - 모든 데코레이터 체인이 정상 작동