아이템 37
'ordinal 인덱싱 대신 EnumMap을 사용하라'

'ordinal 인덱싱 대신 EnumMap을 사용하라'

참고
- Effective Java Item 37
- Java 8 In Action

▸ Enum을 활용하는 방법

1. Matrix 구조의 데이터

2. 열거 타입으로 데이터 관리하기
   - 배열로 관리하기
   - EnumMap으로 관리하기
   - Stream으로 관리하기

3. 정의된 Matrix 구조 데이터

4. 중첩 열거 타입으로 데이터 관리하기
   - 배열로 관리하기
   - Stream & EnumMap으로 관리하기

5. 데이터 타입 추가

Matrix 구조의 데이터 관리

'생애주기(LifeCycle)'에 대한 타입을 세 가지로 정의

사용자의 정원(garden)에 존재하는 식물(plant)을 분류 및 등록

Data: { 생애주기 : 식물 명 }

▸ Enum을 활용하는 방법

1. Matrix 구조의 데이터

2. 열거 타입으로 데이터 관리하기
   - 배열로 관리하기
   - EnumMap으로 관리하기
   - Stream으로 관리하기

3. 정의된 Matrix 구조 데이터

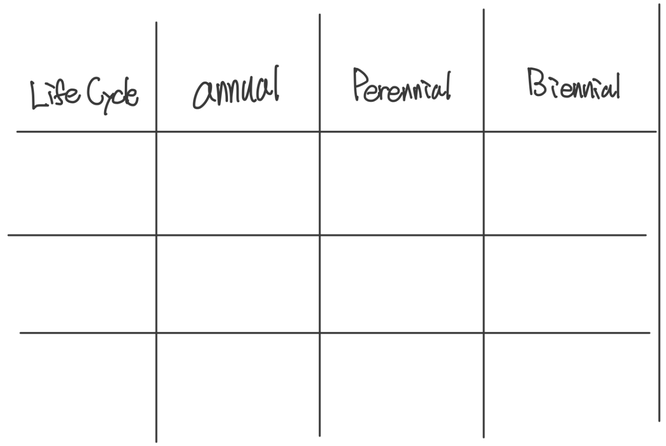
4. 중첩 열거 타입으로 데이터 관리하기
   - 배열로 관리하기
   - Stream & EnumMap으로 관리하기

5. 데이터 타입 추가

Matrix 구조의 데이터 관리

column

row



Java: Set<LifeCycle> sortPlant = new HashSet<>();

▸ Enum을 활용하는 방법

1. Matrix 구조의 데이터

2. 열거 타입으로 데이터 관리하기
   - 배열로 관리하기
   - EnumMap으로 관리하기
   - Stream으로 관리하기

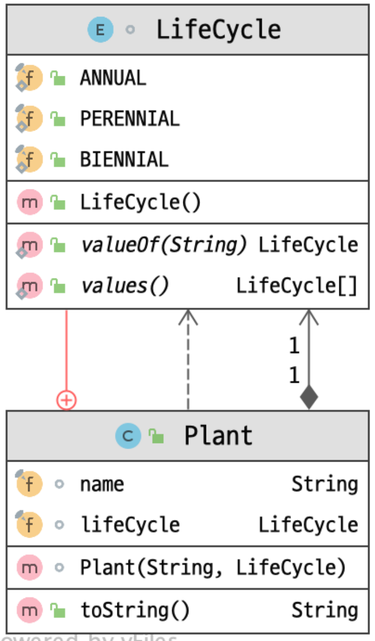
3. 정의된 Matrix 구조 데이터

4. 중첩 열거 타입으로 데이터 관리하기
   - 배열로 관리하기
   - Stream & EnumMap으로 관리하기

5. 데이터 타입 추가

열거 타입으로 관리하기 - 배열

데이터 구조



데이터 활용

```java
class Client {
    public void addPlant(List<Plant> garden) {
        // 식물의 라이프 사이클 종류 정의
        int lifeCycleMaxLength = Plant.LifeCycle.values().length;
        // 1. 정원의 식물들을 라이프 사이클 분류 기준에 따라 식물을 등록
        Set<Plant>[] plantsByLifeCycle = new Set[lifeCycleMaxLength];

        // 2. 배열을 초기화 하기 위해 Set 구현체 생성
        for(int i = 0 ; i < plantsByLifeCycle.length ; i++) {
            plantsByLifeCycle[i] = new HashSet<>();
        }

        // 3. 사용자가 등록한 데이터를 Set 컨테이너에 등록
        for(Plant p : garden) {
            plantsByLifeCycle[p.lifeCycle.ordinal()].add(p);
        }

        // 인덱스의 의미를 알 수 없어 직접 레이블을 달아 데이터 확인 작업 필요
        for(int i = 0 ; i < plantsByLifeCycle.length ; i++) {
            System.out.printf("%s: %s%n", Plant.LifeCycle.values()[i], plantsByLifeCycle[i]);
        }
    }
}
```

배열과 제네릭의 호환되지 않아 **비검사 형변환** 작업 필요

인덱스와 데이터에 대한 **매핑**이 정확하게 되었는지 보장 되어야 함

데이터 조회 시 **argument** 가 **정수** 인지 타입인지 확인 필요

데이터 조회

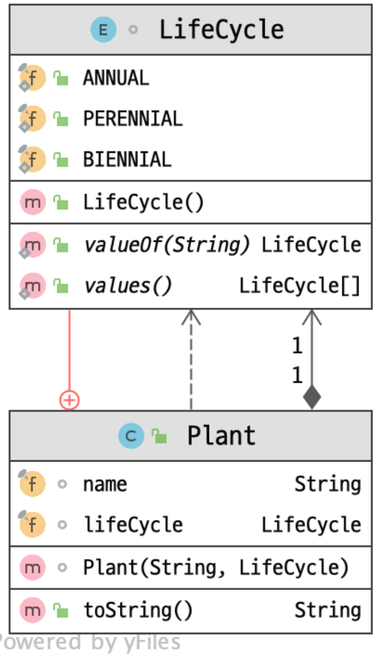
```
ANNUAL: [ANNUAL_TREE_2, ANNUAL_TREE_1, ANNUAL_TREE_3]
PERENNIAL: []
BIENNIAL: [BIENNIAL_TREE_1]
```

▸ Enum을 활용하는 방법

1. Matrix 구조의 데이터
2. 열거 타입으로 데이터 관리하기
   - 배열로 관리하기
   - EnumMap으로 관리하기
   - Stream으로 관리하기
3. 정의된 Matrix 구조 데이터
4. 중첩 열거 타입으로 데이터 관리하기
   - 배열로 관리하기
   - Stream & EnumMap으로 관리하기
5. 데이터 타입 추가

열거 타입으로 관리하기 - EnumMap

데이터 구조



데이터 활용

EnumMap의 생성자는 한정적 타입 토큰을 인자로 받아 런타임 제네릭 타입 정보를 제공받는다.

```
class Client {
    public void addPlantTypeEnumMap(List<Plant> garden) {
        // EnumMap 생성
        Map<Plant.LifeCycle, Set<Plant>> plantByLifeCycle = new EnumMap<>(Plant.LifeCycle.class);

        // EnumMap 내에 Set 컬렉션 생성
        for(Plant.LifeCycle lc : Plant.LifeCycle.values()) {
            plantByLifeCycle.put(lc, new HashSet<>());
        }

        // 사용자가 등록한 식물을 LifeCycle 에 따라 Set을 조회 후 등록
        for(Plant p : garden) {
            plantByLifeCycle.get(p.lifeCycle).add(p);
        }

        System.out.println(plantByLifeCycle);
    }
```

결과 값 조회에 인덱스를 계산하는 과정에 대한 오류가 일어날 일이 없음

데이터의 Key 값인 열거 타입 자체가 문자열을 제공

데이터 조회

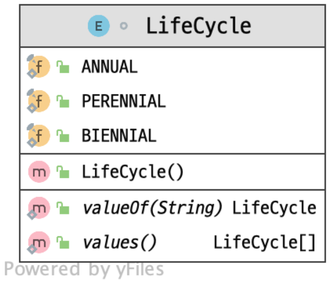
```
{
    ANNUAL=[ANNUAL_TREE_3, ANNUAL_TREE_2, ANNUAL_TREE_1],
    PERENNIAL=[],
    BIENNIAL=[BIENNIAL_TREE_1]
}
```

▸ Enum을 활용하는 방법

1. Matrix 구조의 데이터
2. 열거 타입으로 데이터 관리하기
   - 배열로 관리하기
   - EnumMap으로 관리하기
   - Stream으로 관리하기
3. 정의된 Matrix 구조 데이터
4. 중첩 열거 타입으로 데이터 관리하기
   - 배열로 관리하기
   - Stream & EnumMap으로 관리하기
5. 데이터 타입 추가

Stream으로 관리하기 - EnumMap 미사용

데이터 구조



데이터 활용

```
// {ANNUAL=[ANNUAL_TREE_1, ANNUAL_TREE_2, ANNUAL_TREE_3], BIENNIAL=[BIENNIAL_TREE_1], PERENNIAL=[PERENNIAL_TREE_1]}
@DisplayName("EnumMap을 사용하지 않고 Collections.grouping 테스트")
@Test
void testCase3() {
    Map<Plant.LifeCycle, List<Plant>> garden =
            this.garden.stream().collect(groupingBy(p -> p.lifeCycle));

    List<Plant> excepted = Arrays.asList(
            new Plant( name: "ANNUAL_TREE_1", Plant.LifeCycle.ANNUAL),
            new Plant( name: "ANNUAL_TREE_2", Plant.LifeCycle.ANNUAL),
            new Plant( name: "ANNUAL_TREE_3", Plant.LifeCycle.ANNUAL)
    );

    assertThat(garden).containsEntry(Plant.LifeCycle.ANNUAL, excepted);
}
```

Map 구현체를 직접 사용하는 경우
EnumMap에 비해 공간, 시간적인 효과를 보지 못한다.

EnumMap을 사용하는 경우

LifeCycle에 정의된 모든 타입의 List를 생성한다.

스트림을 사용하는 경우

LifeCycle 필드를 갖고있는 데이터가 존재하는 경우
해당 타입에 대한 List만을 생성한다.

데이터 조회

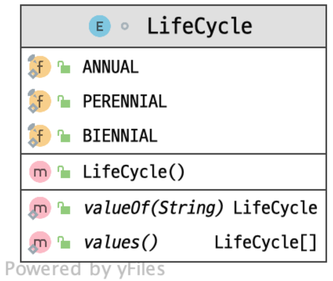
```
{
    BIENNIAL=[BIENNIAL_TREE_1],
    ANNUAL=[ANNUAL_TREE_1, ANNUAL_TREE_2, ANNUAL_TREE_3]
}
```

▸ Enum을 활용하는 방법

1. Matrix 구조의 데이터
2. 열거 타입으로 데이터 관리하기
   - 배열로 관리하기
   - EnumMap으로 관리하기
   - Stream으로 관리하기
3. 정의된 Matrix 구조 데이터
4. 중첩 열거 타입으로 데이터 관리하기
   - 배열로 관리하기
   - Stream & EnumMap으로 관리하기
5. 데이터 타입 추가

Stream으로 관리하기 - EnumMap 사용

데이터 구조



데이터 활용

```
// {ANNUAL=[ANNUAL_TREE_3, ANNUAL_TREE_2, ANNUAL_TREE_1], BIENNIAL=[BIENNIAL_TREE_1]}
@DisplayName("EnumMap을 이용해 데이터와 열거 타입을 매핑하는 테스트")
@Test
void testCase4() {
    Map<Plant.LifeCycle, Set<Plant>> garden = this.garden.stream()
            .collect(
                    groupingBy(
                            p -> p.lifeCycle, () -> new EnumMap<>(Plant.LifeCycle.class), toSet()
                    ));

    Set<Plant> excepted = Sets.newHashSet(
            new Plant( name: "ANNUAL_TREE_1", Plant.LifeCycle.ANNUAL),
            new Plant( name: "ANNUAL_TREE_2", Plant.LifeCycle.ANNUAL),
            new Plant( name: "ANNUAL_TREE_3", Plant.LifeCycle.ANNUAL)
    );

    assertThat(garden).containsEntry(Plant.LifeCycle.ANNUAL, excepted);
}
```

스트림과 EnumMap을 이용하여
데이터와 열거 타입을 매핑

데이터 조회

```
{
    ANNUAL=[ANNUAL_TREE_1, ANNUAL_TREE_2, ANNUAL_TREE_3],
    BIENNIAL=[BIENNIAL_TREE_1]
}
```

▸ Enum을 활용하는 방법

1. Matrix 구조의 데이터
2. 열거 타입으로 데이터 관리하기
   - 배열로 관리하기
   - EnumMap으로 관리하기
   - Stream으로 관리하기
3. 정의된 Matrix 구조 데이터
4. 중첩 열거 타입으로 데이터 관리하기
   - 배열로 관리하기
   - Stream & EnumMap으로 관리하기
5. 데이터 타입 추가

정의된 Matrix 구조 데이터

데이터 구조 정의

	SOLID	LIQUID	GAS
SOLID	**NULL**	MELT	SUBLIME
LIQUID	FREEZE	**NULL**	BOIL
GAS	DEPOSIT	CONDENSE	**NULL**

중첩 열거 타입으로 데이터 관리하기 - 배열

중첩 열거 타입을 배열로 관리하는 경우

	SOLID	LIQUID	GAS
SOLID	**NULL**	MELT	SUBLIME
LIQUID	FREEZE	**NULL**	BOIL
GAS	DEPOSIT	CONDENSE	**NULL**

데이터 구조 정의

```
public enum Phase {
    SOLID, LIQUID, GAS;

    public enum Transition {
        MELT, FREEZE, BOIL,
        CONDENSE, SUBLIME, DEPOSIT;

        private static final Transition[][] TRANSITIONS = {
                {null, MELT, SUBLIME},
                {FREEZE, null, BOIL},
                {DEPOSIT, CONDENSE, null}
        };

        public static Transition from(Phase from, Phase to) {
            return TRANSITIONS[from.ordinal()][to.ordinal()];
        }
    }
}
```

이 차원 배열로 매트릭스를 직접 관리 해야 하는 문제점

컴파일러는 ordinal과 배열 인덱스의 관계를 알 수 없다.
이는 코드 관리 시 코드 내용을 해석할 수 있는 표가 따로 존재해야 한다.

```
class PhaseArrayTest {

    @DisplayName("고체, 액체, 가스에 대한 데이터베이스를 enum으로 작성 및 테스트")
    @Test
    void testCase1() {
        // 행, 열 조회 -> 뭐하는 짓이지?

        assertThat(from(PhaseArray.SOLID, PhaseArray.SOLID)).isNull();
        assertThat(from(PhaseArray.SOLID, PhaseArray.LIQUID)).isEqualTo(PhaseArray.Transition.MELT);
        assertThat(from(PhaseArray.SOLID, PhaseArray.GAS)).isEqualTo(PhaseArray.Transition.SUBLIME);
        assertThat(from(PhaseArray.SOLID, PhaseArray.PLASMA)).isNull();
```

고체에서 액체로 변화하는 현상 -> 용해(Melt)

▸ Enum을 활용하는 방법

1. Matrix 구조의 데이터
2. 열거 타입으로 데이터 관리하기
   - 배열로 관리하기
   - EnumMap으로 관리하기
   - Stream으로 관리하기
3. 정의된 Matrix 구조 데이터
4. 중첩 열거 타입으로 데이터 관리하기
   - 배열로 관리하기
   - Stream & EnumMap으로 관리하기
5. 데이터 타입 추가

중첩 열거 타입으로 데이터 관리하기 - Stream & EnumMap

열거 타입을 '이전' '이후' 쌍으로 연결한 타입을 정의

```
public enum PhaseEnumMap {
    SOLID, LIQUID, GAS;

    public enum Transition {
        MELT(SOLID, LIQUID), FREEZE(LIQUID, SOLID),
        BOIL(LIQUID, GAS), CONDENSE(GAS, LIQUID),
        SUBLIME(SOLID, GAS), DEPOSIT(GAS, SOLID);

        private final PhaseEnumMap from;
        private final PhaseEnumMap to;

        Transition(PhaseEnumMap from, PhaseEnumMap to) {
            this.from = from;
            this.to = to;
        }

        private static final Map<PhaseEnumMap, Map<PhaseEnumMap, Transition>> m =
                Stream.of(values()) // enum 타입 두 개를 매핑한 필드 리스트
                        .collect(
                                groupingBy(
                                        t -> t.from, () -> new EnumMap<>(PhaseEnumMap.class),
                                        toMap(
                                                t -> t.to,
                                                t -> t,
                                                (x, y) -> y,
                                                () -> new EnumMap<>(PhaseEnumMap.class)
                                        ))
                        );

        public static Transition from(PhaseEnumMap from, PhaseEnumMap to) { return m.get(from).get(to); }
    }
}
```

열거 타입을 쌍으로
from -> to에 대한 결과 값을
필드로 정의

```
toMap(
    KeyMapper,
    ValueMapper,
    mergeFunction,
    mapSupplier
);
```

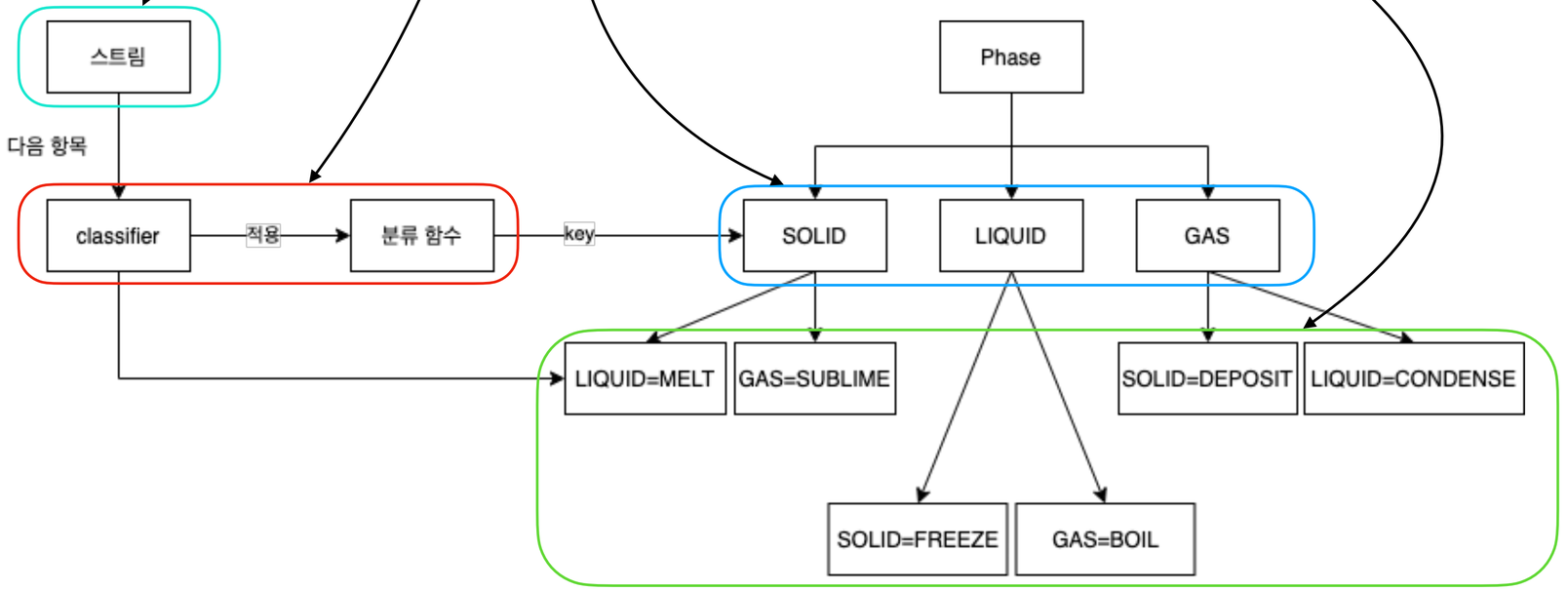
```
class PhaseEnumMapTest {

    @DisplayName("EnumMap 으로 구성된 데이터 관리 테스트 ")
    @Test
    void testCase1() {

        assertThat(PhaseEnumMap.Transition.from(PhaseEnumMap.SOLID, PhaseEnumMap.SOLID)).isNull();
        assertThat(PhaseEnumMap.Transition.from(PhaseEnumMap.SOLID, PhaseEnumMap.LIQUID)).isEqualTo(PhaseEnumMap.Transition.MELT);
        assertThat(PhaseEnumMap.Transition.from(PhaseEnumMap.SOLID, PhaseEnumMap.GAS)).isEqualTo(PhaseEnumMap.Transition.SUBLIME);
        assertThat(PhaseEnumMap.Transition.from(PhaseEnumMap.SOLID, PhaseEnumMap.PLASMA)).isNull();
```

▸ Enum을 활용하는 방법

1. Matrix 구조의 데이터
2. 열거 타입으로 데이터 관리하기
   - 배열로 관리하기
   - EnumMap으로 관리하기
   - Stream으로 관리하기
3. 정의된 Matrix 구조 데이터
4. 중첩 열거 타입으로 데이터 관리하기
   - 배열로 관리하기
   - Stream & EnumMap으로 관리하기
5. 데이터 타입 추가

Stream & EnumMap

```
private static final Map<PhaseEnumMap, Map<PhaseEnumMap, Transition>> m =
        Stream.of(values()) // enum 타입 두 개를 매핑한 필드 리스트
                .collect(groupingBy(
                        t -> t.from, // Phase 타입을 Key 값으로 정의
                        () -> new EnumMap<>(PhaseEnumMap.class), // Value를 Map 타입으로 정의
                        toMap(
                                t -> t.to,
                                t -> t,
                                (x, y) -> y,
                                () -> new EnumMap<>(PhaseEnumMap.class)
                        )));
```



▸ Enum을 활용하는 방법

1. Matrix 구조의 데이터

2. 열거 타입으로 데이터 관리하기
   - 배열로 관리하기
   - EnumMap으로 관리하기
   - Stream으로 관리하기

3. 정의된 Matrix 구조 데이터

4. 중첩 열거 타입으로 데이터 관리하기
   - 배열로 관리하기
   - Stream & EnumMap으로 관리하기

5. 데이터 타입 추가

데이터 타입 추가

중첩 열거 타입을 배열로 관리하는 경우 - 타입 추가 시

	SOLID	LIQUID	GAS	PLASMA
SOLID	NULL	MELT	SUBLIME	NULL
LIQUID	FREEZE	NULL	BOIL	NULL
GAS	DEPOSIT	CONDENSE	NULL	IONIZE
PLASMA	NULL	NULL	DEIONIZE	NULL

배열 기반의 데이터 구조 정의

```java
public enum PhaseArray {
    SOLID, LIQUID,
    GAS, PLASMA;

    public enum Transition {
        MELT, FREEZE, BOIL, CONDENSE,
        SUBLIME, DEPOSIT, IONIZE, DEIONIZE;

        private static final Transition[][] TRANSITIONS = {
                {null, MELT, SUBLIME, null}, // SOLID
                {FREEZE, null, BOIL, null}, // LIQUID
                {DEPOSIT, CONDENSE, null, IONIZE}, // GAS
                {null, null, DEIONIZE, null} // PLASMA
        };

        public static Transition from(PhaseArray from, PhaseArray to) {
            return TRANSITIONS[from.ordinal()][to.ordinal()];
        }
    }
}
```

▸ Enum을 활용하는 방법

1. Matrix 구조의 데이터
2. 열거 타입으로 데이터 관리하기
   - 배열로 관리하기
   - EnumMap으로 관리하기
   - Stream으로 관리하기
3. 정의된 Matrix 구조 데이터
4. 중첩 열거 타입으로 데이터 관리하기
   - 배열로 관리하기
   - Stream & EnumMap으로 관리하기
5. 데이터 타입 추가

데이터 타입 추가

열거 타입을 '이전' '이후' 쌍으로 연결한 타입을 정의 - 타입 추가

```
public enum PhaseEnumMap {
    SOLID, LIQUID, GAS, PLASMA;

    public enum Transition {
        MELT(SOLID, LIQUID), FREEZE(LIQUID, SOLID),
        BOIL(LIQUID, GAS), CONDENSE(GAS, LIQUID),
        SUBLIME(SOLID, GAS), DEPOSIT(GAS, SOLID),
        IONIZE(GAS, PLASMA), DEIONIZE(PLASMA, GAS);

        private final PhaseEnumMap from;
        private final PhaseEnumMap to;

        Transition(PhaseEnumMap from, PhaseEnumMap to) {
            this.from = from;
            this.to = to;
        }

        private static final Map<PhaseEnumMap, Map<PhaseEnumMap, Transition>> m =
                Stream.of(values()) // enum 타입 두 개를 매핑한 필드 리스트
                        .collect(groupingBy(
                                t -> t.from, // Phase 타입을 Key 값으로 정의
                                () -> new EnumMap<>(PhaseEnumMap.class), // Value를 Map 타입으로 정의
                                toMap(
                                        t -> t.to,
                                        t -> t,
                                        (x, y) -> y,
                                        () -> new EnumMap<>(PhaseEnumMap.class)
                                )));

        public static Transition from(PhaseEnumMap from, PhaseEnumMap to) { return m.get(from).get(to); }
    }
}
```

하나의 기준 타입과 두 가지 현상에 대한 타입을 추가

- Enum을 활용하는 방법

1. Matrix 구조의 데이터
2. 열거 타입으로 데이터 관리하기
   - 배열로 관리하기
   - EnumMap으로 관리하기
   - Stream으로 관리하기
3. 정의된 Matrix 구조 데이터
4. 중첩 열거 타입으로 데이터 관리하기
   - 배열로 관리하기
   - Stream & EnumMap으로 관리하기
5. 데이터 타입 추가

▸ Enum을 활용하는 방법

1. Matrix 구조의 데이터

2. 열거 타입으로 데이터 관리하기
   - 배열로 관리하기
   - EnumMap으로 관리하기
   - Stream으로 관리하기

3. 정의된 Matrix 구조 데이터

4. 중첩 열거 타입으로 데이터 관리하기
   - 배열로 관리하기
   - Stream & EnumMap으로 관리하기

5. 데이터 타입 추가 리펙토링