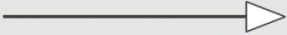







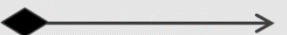


클래스 다이어그램을 이용한 관계 표현

관계	UML 표기
Generalization (일반화)	
Realization (실체화)	
Dependency (의존)	
Association (연관)	
Directed Association (직접연관)	
Aggregation (집합, 집합연관)	
	
Composition (합성, 복합연관)	
	

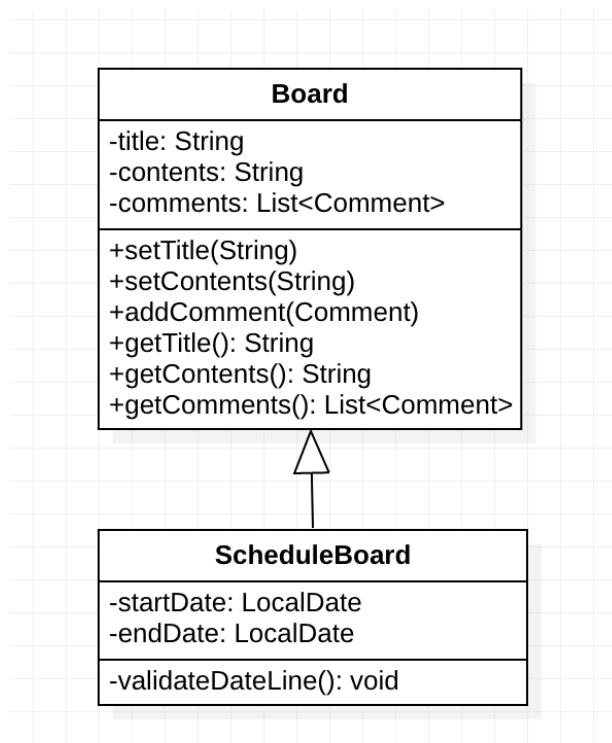
[그림 6] 클래스 관계 종류

참조 : <http://www.nextree.co.kr/p6753/>

클래스간의 연관관계는 위 7가지로 나타낼 수 있습니다. 각각이 어떤 관계를 의미하는지, 그리고 클래스 다이어그램으로 어떻게 표현되는지 알아보도록 하겠습니다.

Generalization (일반화)

Generalization은 우리가 일반적으로 알고 있는 상속을 의미합니다. 이는 클래스 다이어그램에서는 **실선에 비어있는 화살표**로 표시합니다.



이러한 도메인 모델을 코드로 구현하면 아래와 같습니다.(get, set 메서드는 생략하도록 하겠습니다.)

```
public class Board {

    private String title;

    private String contents;

    private List<Comment> comments;

    // 기본 getter / setter 메서드들

}
```

```
class SchedulerBoard extends Board {

    private LocalDate startDate;

    private LocalDate endDate;

    public void validateDateLine() {

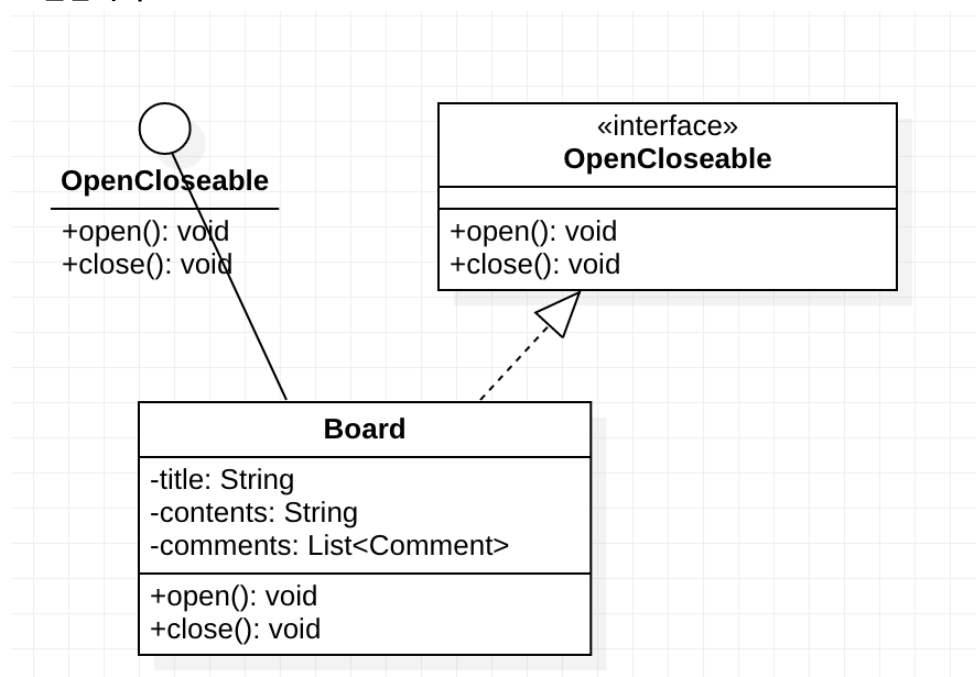
        ...

    }

}
```

Realization (실체화)

Realization은 interface에 있는 spec을 오버라이딩하여 실제로 구현하는 것을 말합니다. **Realization**은 점선과 비어있는 화살표로 표현합니다.



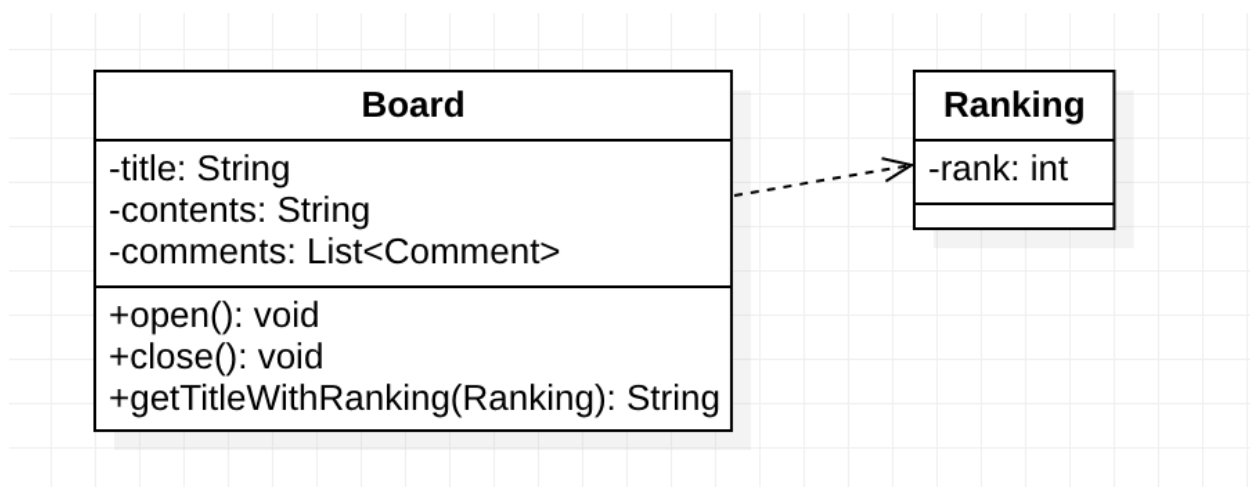
위 다이어그램의 오른쪽과 왼쪽은 모두 동일한 의미를 가집니다. 코드로 구현하면 아래와 같습니다.(Board의

attribute는 구현 생략하였습니다.)

```
public interface OpenCloseable {  
  
    void open();  
  
    void close();  
  
}  
  
public class Board implements OpenCloseable {  
  
    public void open() {  
  
        System.out.println("open method");  
  
    }  
  
    public void close() {  
  
        System.out.println("close method");  
  
    }  
  
}
```

Dependency (의존)

Dependency는 클래스간의 참조가 일어나는 것 중 하나입니다. Dependency 참조는 메서드 내에서 대상 클래스의 객체를 생성하거나 사용, 리턴받아 사용하는 것을 말합니다. 그리고 이 참조는 해당 클래스와의 관계를 계속 유지하지 않습니다. dependency는 점선과 화살표로 이루어져 있습니다.



코드로 표현하면 아래와 같습니다.

```
public class Board {
```

```

private String title;

public String getTitleWithRanking(Ranking ranking) {

    return title + ranking.getRanking();

}

}

```

```

public class Ranking {

    private int rank;

    public int getRank() {

        return rank;

    }

}

```

Dependency는 메서드의 호출이 끝나면 연관된 클래스와의 관계가 마무리된다는 것이 중요합니다.

Association & Direct Association (연관)

클래스 다이어그램에서의 **Association**은 다른 객체의 참조를 가지는 필드를 의미합니다. 실선으로 표현되며 방향성이 존재하는 경우에는 화살표를 넣어 표시합니다. 그리고 둘의 연관관계가 어떻게 되는지 숫자로 표시할 수 있습니다.

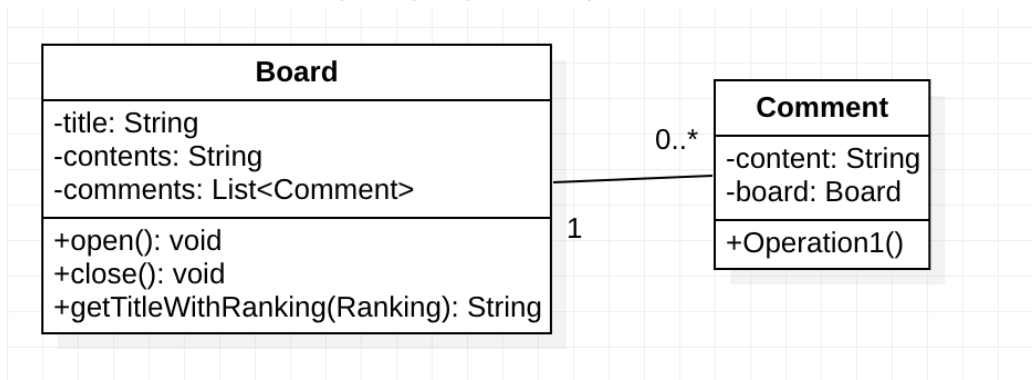
연관계의 숫자 표현은 아래와 같이 합니다.

1 - 1개의 표현

* - 0 ~ n 개의 표현

n...m : n 부터 m까지 연관관계를 맺음

아래 예제는 양방향 연관관계를 가지며 1(Board) : n(Comment)의 관계를 표시한 것입니다.



아래의 코드는 위의 이미지를 코드로 구현한 것입니다.

```

public class Board {

```

```

private List<Comment> comments;

public void addComment(Comment comment) {

    comments.add(comment);

}

}

public class Comment {

    private Board board;

    public void setBoard(Board board) {

        this.board = board;

        board.addComment(this);

    }

}

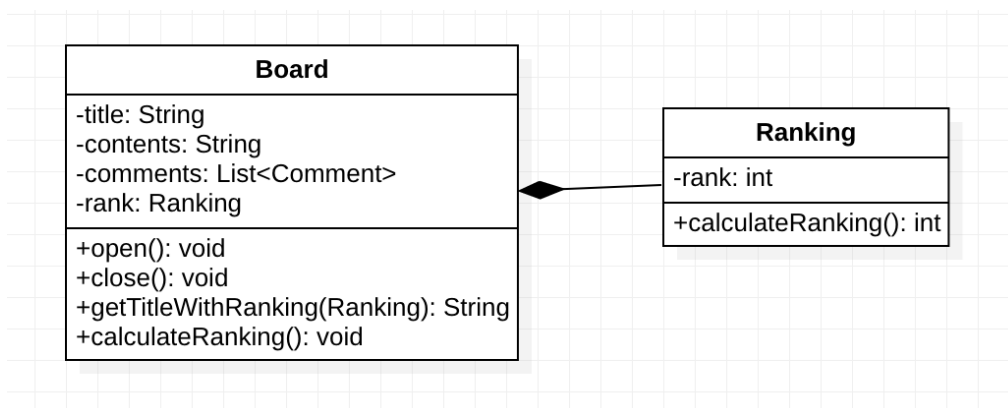
```

Aggregation (집합) & Composition (합성)

클래스 다이어그램에서 Aggregation과 Composition은 Association의 특수한 관계입니다. **Aggregation은 Association의 집합관계를 나타내는 것으로 Collection이나 Array를 이용하는 관계**입니다. 하지만 이 관계는 일반적인 Association으로도 충분히 나타낼 수 있는 관계입니다. 우리가 위에서 Association의 예제가 바로 1 : N 연관관계를 나타낸 것입니다. Aggregation과 Association의 모호성은 지속적으로 논란이 되고 있지만 결론이 나지 않았다고 합니다.

Aggregation은 실선에 빈 다이아몬드로 나타냅니다.

Composition은 클래스 연관관계에서 강한 결합의 관계를 의미합니다. 즉, 참조하는 클래스의 라이프 사이클이 종속적이라는 말입니다. 다이어그램으로는 실선에 채워져있는 다이아몬드로 표시합니다.



조금 억지스럽기는 하지만 rank class Board에 종속적인 상황입니다. 코드는 아래와 같습니다.

```

public class Board {

```

```
private Ranking rank;

public Board() {

    rank = new Rank();

}

public void calculateRanking() {

    System.out.println(rank.calculateRanking());

}

}
```

```
public class Ranking {

    private int rank;

    public int calculateRanking() {

        ... [logic] ...

        return rank;

    }

}
```