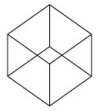
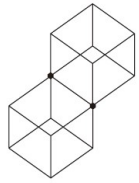
9. 옵서버 패턴



JAVA 객체 지향 디자인 패턴

UML과 GoF 디자인 패턴 핵심 10가지로 배우는



학습목표

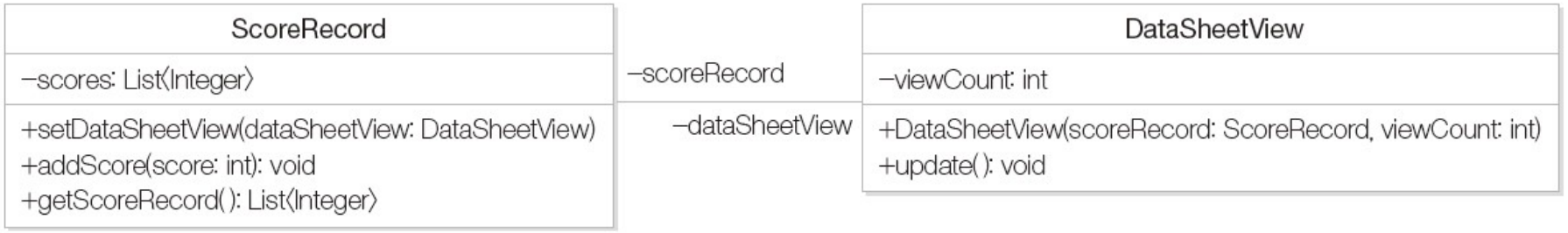
학습목표

- 데이터의 변화를 통보하는 방법 이해하기
- 옵서버 패턴을 통한 통보의 캡슐화 방법 이해하기
- 사례 연구를 통한 옵서버 패턴의 핵심 특징 이해하기

9.1 [추후확장가능성] 여러 가지 방식으로 성적 출력하기

- 요구사항 : 입력된 성적 값을 출력하는 프로그램 코드 작성
- [일차적으로 개발된] 성적 출력 프로그램
  - ScoreRecord 클래스: 점수를 저장/관리하는 클래스
  - DataSheetView 클래스: 점수를 목록형태로 출력하는 클래스

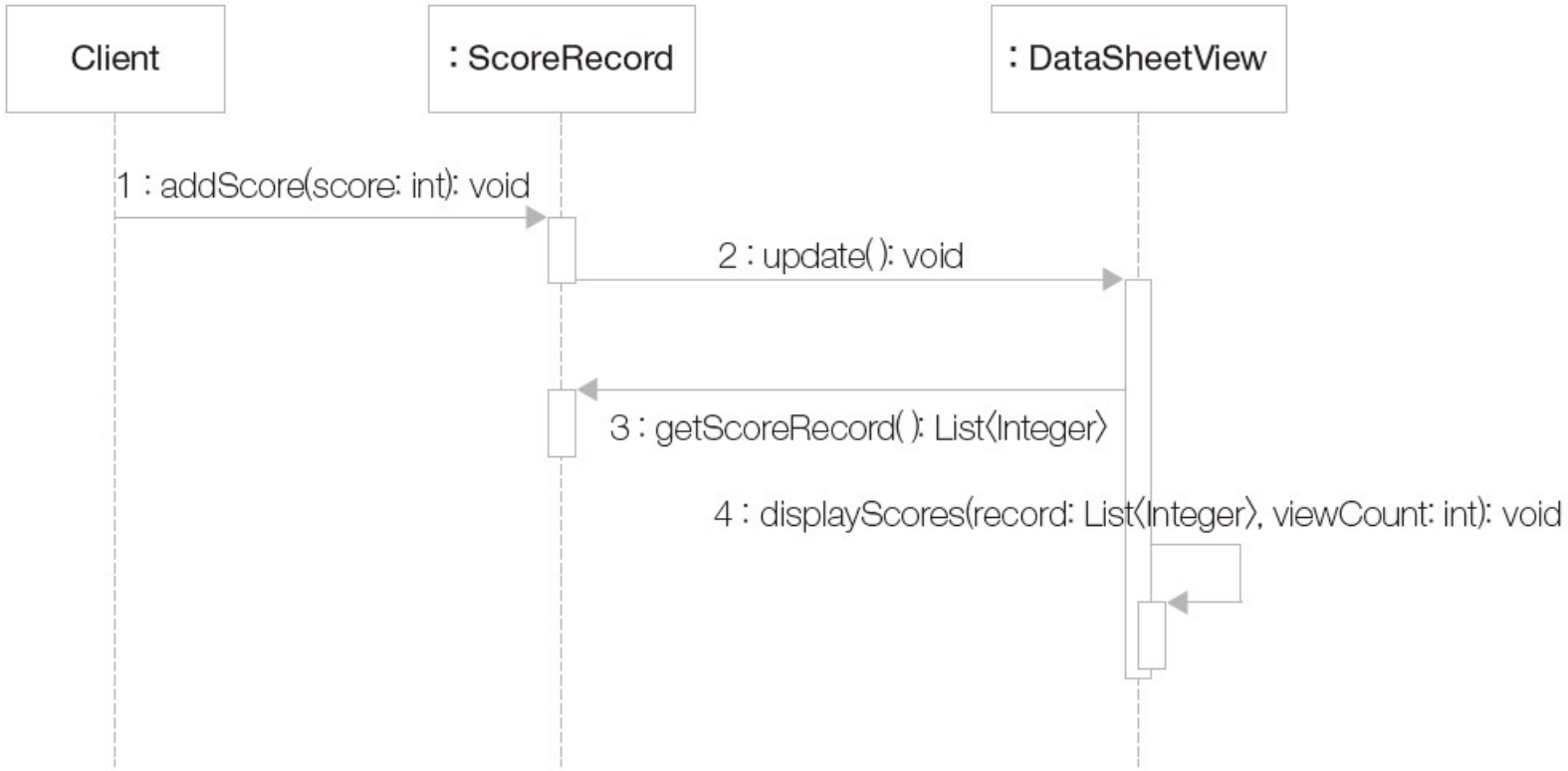
그림 9-1 ScoreRecord 클래스의 값을 출력하는 DataSheetView 클래스의 설계



9.1 여러 가지 방식으로 성적 출력하기

❖ 성적 출력 프로그램: 순차 다이어그램

그림 9-2 ScoreRecord와 DataSheetView 클래스 사이의 상호작용



소스 코드

코드 9-1

```
public class ScoreRecord {
  private List<Integer> scores = new ArrayList<Integer>() ;  // 점수를 저장함
  private DataSheetView dataSheetView ;  // 목록 형태로 점수를 출력하는 클래스

  public void setDataSheetView(DataSheetView dataSheetView) {
    this.dataSheetView = dataSheetView ;
  }
  public void addScore(int score) {  // 새로운 점수를 축함
    scores.add(score) ;  // scores 목록에 주어진 점수를 추가함
    dataSheetView.update() ;  // scores가 변경됨을 통보함
  }
  public List<Integer> getScoreRecord() {
    return scores ;
  }
}
```

소스 코드

코드 9-1

```
public class DataSheetView {
   private ScoreRecord scoreRecord ;
   private int viewCount ;

   public DataSheetView(ScoreRecord scoreRecord, int viewCount) {
      this.scoreRecord = scoreRecord ;
      this.viewCount = viewCount ;
   }
   public void update() {  // 점수의 변경을 통보 받음
      List<Integer> record = scoreRecord.getScoreRecord() ; // 점수를 조회함
      displayScores(record, viewCount);  // 조회된 점수를 viewCount만큼 출력함
   }
   private void displayScores(List<Integer> record, int viewCount) {
      System.out.print("List of " + viewCount + " entries: ") ;
      for ( int i = 0 ; i < viewCount && i < record.size() ; i ++ ) {
         System.out.print(record.get(i) + " ") ;
      }
      System.out.println() ;
   }
}
```

소스 코드

코드 9-1

```
public class Client {
   public static void main(String[] args) {
      ScoreRecord scoreRecord = new ScoreRecord() ;
      // 3개까지의 점수만 출력함
      DataSheetView dataSheetView = new DataSheetView(scoreRecord, 3) ;

      scoreRecord.setDataSheetView(dataSheetView) ;

      for (int index = 1 ; index <= 5 ; index ++ ) {
         int score = index * 10 ;
         System.out.println("Adding " + score) ;
         // 10 20 30 40 50을 추가함, 추가할 때마다 최대 3개의 점수만 출력함
         scoreRecord.addScore(score) ;
      }
   }
}
```

9.2 추가요구사항&문제점

❖ 추가요구사항

- 성적을 다른 방식으로 출력하고 싶다면 어떤 변경 작업을 해야 하는가? 예를 들어 성적을 목록으로 출력하지 않고 최소/최대값만을 출력하려면?
- 뿐만 아니라 성적을 동시에 여러 가지 형태로 출력하려면 어떤 변경 작업을 해야 하는가? 예를 들어 성적이 입력되면 최대 3개 목록으로 출력, 최대 5개 목록으로 출력 그리고 동시에 최소/최대값만을 출력하려면?
- 그리고 프로그램이 실행 시에 성적의 출력 대상이 변경되는 것을 지원한다면 어떤 변경 작업을 해야 하는가? 예를 들어 처음에는 목록으로 출력하고 나중에는 최소/최대값을 출력하려면?

9.2.1 성적을 다른 형태로 출력하는 경우

❖ 추가요구사항 – 점수가 입력되면, 점수 목록을 출력하는 대신 최소/최대 값만 출력하게 하려면??

코드 9-2

```
public class MinMaxView {  // 전체 점수가 아니라 최소/최대값만을 출력하는 클래스
  private ScoreRecord scoreRecord ;

  public MinMaxView(ScoreRecord scoreRecord) {
    this.scoreRecord = scoreRecord ;
  }
  public void update() {
    List<Integer> record = scoreRecord.getScoreRecord() ;
    displayMinMax(record);  // 최소/최대값만을 출력
  }
  private void displayMinMax(List<Integer> record) {
    int min =  Collections.min(record, null) ;
    int max =  Collections.max(record, null) ;
    System.out.println("Min: " + min + " Max: " + max) ;
  }
}
```

코드 9-2

```
public class ScoreRecord {
  private List<Integer> scores = new ArrayList<Integer>() ;
  private MinMaxView minMaxView ;

  public void setStatisticsView(MinMaxView minMaxView) {  // MinMaxView를 설정함
    this.minMaxView = minMaxView;
  }
  public void addScore(int score) {
    scores.add(score) ;
    minMaxView.update() ;  // MinMaxView에게 점수의 변경을 통보함
  }
  public List<Integer> getScoreRecord() {
    return scores ;
  }
}
```

- MinMaxView를 이용하도록 소스코드가 수정되었음
- 기능 변경을 위해서 기존 소스 코드를 수정하므로 OCP를 위반하는 것임

코드 9-2

```
public class Client {
  public static void main(String[] args) {
    ScoreRecord scoreRecord = new ScoreRecord() ;
    MinMaxView minMaxView = new MinMaxView(scoreRecord) ;

    scoreRecord.setMinMaxView(minMaxView) ;

    for (int index = 1 ; index <= 5 ; index ++ ) {
      int score = index * 10 ;
      System.out.println("Adding " + score) ;
      // 10 20 30 40 50을 추가함, 추가할 때마다 최소/최대 점수만 출력함
      scoreRecord.addScore(score) ;
    }
  }
}
```

9.2.2 동시에 여러 가지 방식으로 성적을 출력하는 경우

- 추가요구사항
  - 예) 성적이 입력되었을때, 최대 3개 목록, 최대 5개 목록, 최소/최대 값을 동시에 출력하게 함
  - 예) 처음에는 목록으로 출력하고 나중에는 최소/최대 값을 출력하게 함
  - ➔ 실제 출력 기능을 고려하기 전에, 점수가 입력되면 복수개의 대상 클래스를 갱신하는 구조를 먼저 생각하라
  - 목록으로 출력하는 것 : DataSheetView Class
  - 최소/최대 값을 출력하는 것 : MinMaxView Class
- 코드 9-3. 2개의 DataSheetView 객체와 1개의 MinMaxView에 성적 추가를 통보할수 있도록 변경된 ScoreRecord 클래스

코드 9-3

```
public class ScoreRecord {
   private List<Integer> scores = new ArrayList<Integer>() ;
   private List<DataSheetView> dataSheetViews = new ArrayList<DataSheetView>() ;
   private MainMaxView minMaxView ;
   public void addDataSheetView(DataSheetView dataSheetView) {
      dataSheetViews.add(dataSheetView) ;
   }
   public void setMinMaxView(MainMaxView minMaxView) {
      this.minMaxView = minMaxView ;
   }
   public void addScore(int score) {
      scores.add(score) ;
      for ( DataSheetView dataSheetView: dataSheetViews )
         dataSheetView.update() ;  // 각 DataSheetView에게 점수의 변경을 통보
      minMaxView.update() ; // MinMaxView에게 점수의 변경을 통보
   }
   public List<Integer> getScoreRecord() { return scores ; }
}
// DataSheetView 클래스는 코드 9-1과 동일
// MinMaxView 클래스는 코드 9-2와 동일
```

- 기능 변경을 위해서 기존 소스 코드를 수정하므로 OCP를 위반하는 것임

코드 9-3

```
public class Client {
   public static void main(String[] args) {
      ScoreRecord scoreRecord = new ScoreRecord() ;
      // 3개 목록의 DataSheetView 생성
      DataSheetView dataSheetView3 = new DataSheetView(scoreRecord, 3) ;
      // 5개 목록의 DataSheetView 생성
      DataSheetView dataSheetView5 = new DataSheetView(scoreRecord, 5) ;
      MainMaxView minMaxView = new MainMaxView(scoreRecord) ;

      scoreRecord.addDataSheetView(dataSheetView3) ;
      scoreRecord.addDataSheetView(dataSheetView5) ;
      scoreRecord.setMinMaxView(minMaxView) ;

      for (int index = 1 ; index <= 5 ; index ++ ) {
         int score = index * 10 ;
         System.out.println("Adding " + score) ;
         // 10 20 30 40 50을 추가함
         // 추가할 때마다 최대 3개목록, 최대 5개 목록, 그리고 최소/최대 점수가 출력됨
         scoreRecord.addScore(score) ;
      }
   }
}
```

9.3. 해결책

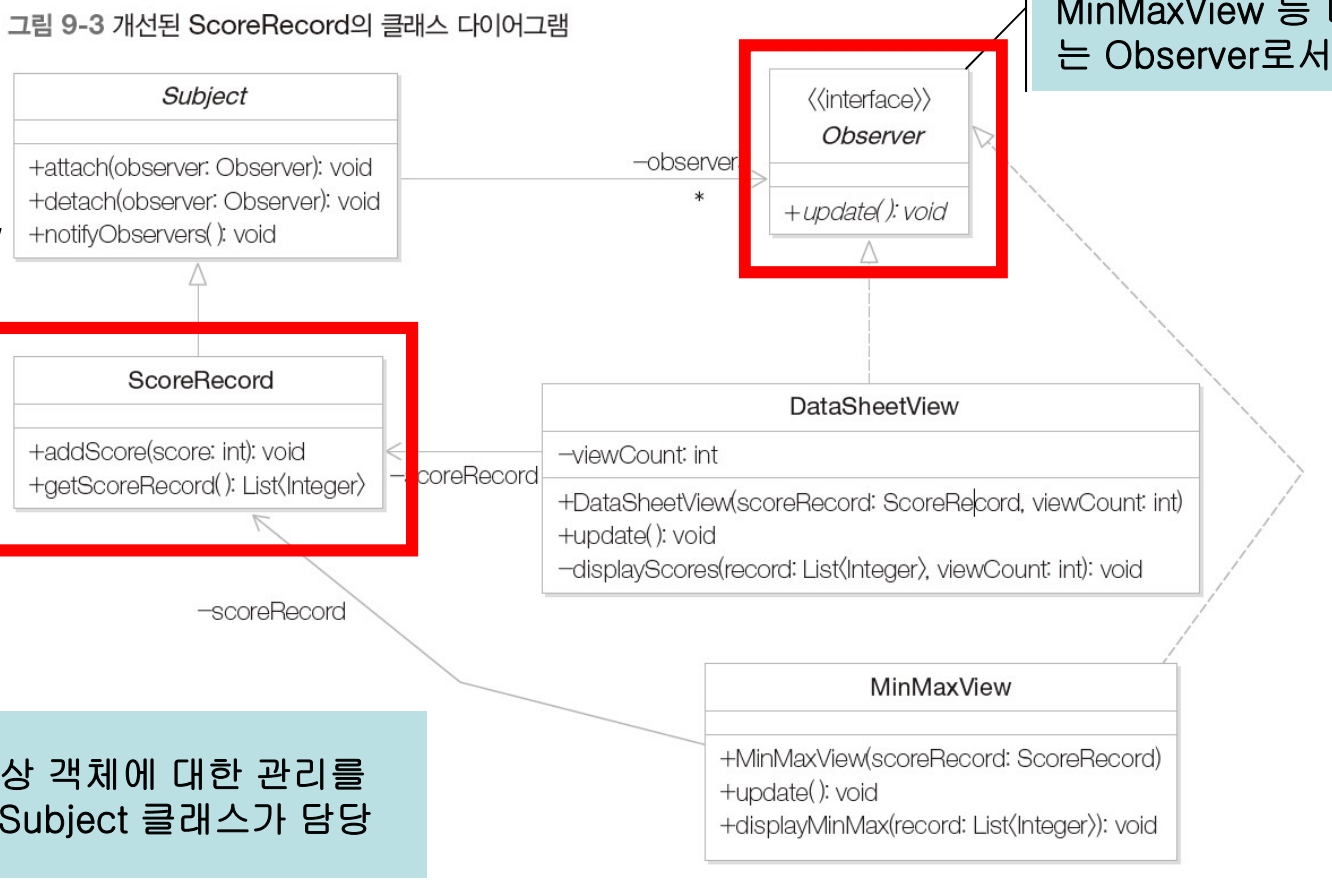
- 성적 통보 대상이 변경되더라도 ScoreRecord 클래스를 그대로 재사용할 수 있어야 함
  - ScoreRecord 클래스에서 변화되는 부분을 식별하고 이를 일반화시켜야 함

- ScoreRecord Class
  - addScore에서 각 출력 객체의 update method 호출
  - 모든 출력객체에 대하여 이루어지는 동일한 기능
  - 이를 일반화 (상위 객체로)

```
public void addScore(int score) {
  scores.add(score) ;
  for ( DataSheetView dataSheetView: dataSheetViews )
    dataSheetView.update() ;  // 각 DataSheetView에게 점수의 변경을 통보
  minMaxView.update() ; // MinMaxView에게 점수의 변경을 통보
}
```

OCP를 위반하는 것임

그림 9-3 개선된 ScoreRecord의 클래스 다이어그램



DataSheetView, MinMaxView 등 대상 클래스는 Observer로서 구현

통보 대상 객체에 대한 관리를 별도의 Subject 클래스가 담당

❖ Abstract class Subject

- 성적 변경에 관심이 있는 대상 객체를 관리함
- 성적 변경에 관심이 있는 대상 객체를 추가하거나 제거
-

9.3. 해결책: 소스 코드

코드 9-4

```
public interface Observer {  // 추상화된 통보 대상
  abstract public void update() ; //데이터의 변경을 통보했을때 처리하는 메서드
}
```

```
public abstract class Subject { // 추상화된 변경 관심 대상 데이터
  private List<Observer> observers = new ArrayList<Observer>() ;
  public void attach(Observer observer) { // 옵서버 즉 통보 대상을 추가함
    observers.add(observer) ;
  }
  public void detach(Observer observer) { // 옵서버 즉 통보 대상을 제거함
    observers.remove(observer) ;
  }
  // 통보 대상 목록, 즉 observers의 각 옵서버에게 변경을 통보함
  public void notifyObservers() {
    for ( Observer o : observers ) o.update() ;
  }
}
```

9.3. 해결책: 소스 코드

코드 9-4

```
public class ScoreRecord extends Subject {  // 구체적인 변경 감시 대상 데이터
  private List<Integer> scores = new ArrayList<Integer>() ;
  public void addScore(int score) {
    scores.add(score) ;
    // 데이터가 변경되면 Subject 클래스의 notifyObservers 메서드를 호출해
    // 각 옵서버(통보 대상 객체)에게 데이터의 변경을 통보함
    notifyObservers() ;
  }
  public List<Integer> getScoreRecord() {
    return scores ;
  }
}
```

```
// DataSheetView는 Observer의 기능 즉 update 메서드를 구현함으로써 통보 대상이 됨
public class DataSheetView implements Observer {
// 코드 9-1과 동일
}
```

```
// MinMaxView는 Observer의 기능 즉 update 메서드를 구현함으로써 통보 대상이 됨
public class MinMaxView implements Observer {
// 코드 9-2과 동일
}
```

9.3. 해결책: 소스 코드

코드 9-4

```
public class Client {
  public static void main(String[] args) {
    ScoreRecord scoreRecord = new ScoreRecord() ;
    DataSheetView dataSheetView3 = new DataSheetView(scoreRecord, 3) ;
    DataSheetView dataSheetView5 = new DataSheetView(scoreRecord, 5) ;
    MinMaxView minMaxView = new MinMaxView(scoreRecord) ;
    // 3개 목록 DataSheetView를 ScoreRecord에 Observer로 추가함
    scoreRecord.attach(dataSheetView3) ;
    // 5개 목록 DataSheetView를 ScoreRecord에 Observer로 추가함
    scoreRecord.attach(dataSheetView5) ;
    // MinMaxView를 ScoreRecord에 Observer로 추가함
    scoreRecord.attach(minMaxView) ;

    for (int index = 1 ; index <= 5 ; index ++ ) {
      int score = index * 10 ;
      System.out.println("Adding " + score) ;
      scoreRecord.addScore(score) ;
    }
  }
}
```

❖ 또다른 변경사항

- 처음 5개의 점수는 목록(DataSheetView 클래스) 출력과 함께 최소/최대 값(MinMaxView 클래스)를 출력하고 이후 5개 점수는 최소/최대값 출력과 함께 합계/평균을 출력하는 프로그램을 작성하라

- 합계/평균 출력 --> StatisticsVIew 클래스

- 포인트 : ScorRecord 클래스는 수정하지 않음

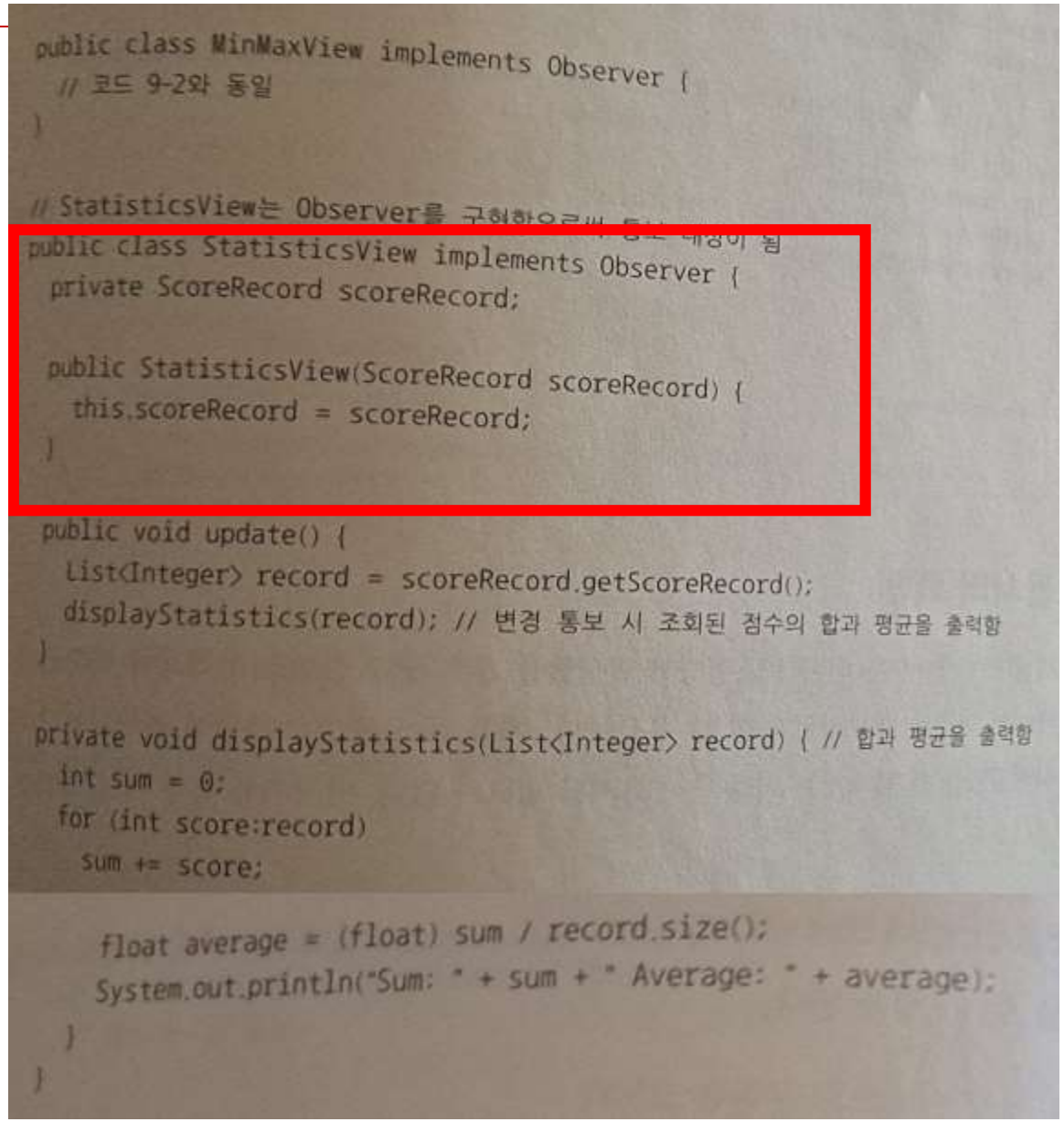
코드 9-5

```
public interface Observer {
  public abstract void update();
}

public abstract class Subject {
  // 코드 9-4와 동일
}

public class ScoreRecord extends Subject {
  // 코드 9-4와 동일
}

public class DataSheetView implements Observer {
  // 코드 9-1과 동일
}
```



```
public class MinMaxView implements Observer {
  // 코드 9-2와 동일
}

// StatisticsView는 Observer를 구현함으로써 통보 대상이 됨
public class StatisticsView implements Observer {
 private ScoreRecord scoreRecord;

 public StatisticsView(ScoreRecord scoreRecord) {
   this.scoreRecord = scoreRecord;
 }

 public void update() {
   List<Integer> record = scoreRecord.getScoreRecord();
   displayStatistics(record); // 변경 통보 시 조회된 점수의 합과 평균을 출력함
 }

 private void displayStatistics(List<Integer> record) { // 합과 평균을 출력함
   int sum = 0;
   for (int score:record)
     sum += score;
     float average = (float) sum / record.size();
     System.out.println("Sum: " + sum + " Average: " + average);
   }
 }
```

```
public class Client {
  public static void main(String[] args) {
    ScoreRecord scoreRecord = new ScoreRecord();
    DataSheetView dataSheetView3 = new DataSheetView(scoreRecord, 3);
    scoreRecord.attach(dataSheetView3);
    MinMaxView minMaxView = new MinMaxView(scoreRecord);
    scoreRecord.attach(minMaxView);

    // 3개 목록 DataSheetView, 5개 목록 DataSheetView, 그리고 MinMaxView가 Observer로 설정됨
    for (int index = 1; index <= 5; index++) {
      int score = index * 10;
      System.out.println("Adding " + score);
      scoreRecord.addScore(score); // 각 점수 추가 시 최대 3개 목록, 5개 목록, 최소/최대 값을 출력함
    }

    scoreRecord.detach(dataSheetView3); // 3개 목록 DataSheetView는 이제 Observer가 아님
    StatisticsView statisticsView = new StatisticsView(scoreRecord);
    scoreRecord.attach(statisticsView); // StatisticsView가 Observer로서 설정됨

    // 이제 5개 목록 DataSheetView, MinMaxView, 그리고 StatisticsView가 Observer임
    for (int index = 1; index <= 5; index++) {
      int score = index * 10;
      System.out.println("Adding " + score);
      scoreRecord.addScore(score); // 각 점수 추가 시 최대 5개 목록, 최소/최대 값, 합/평균을 출력함
    }
  }
}
```

9.4 옵서버(observer) 패턴

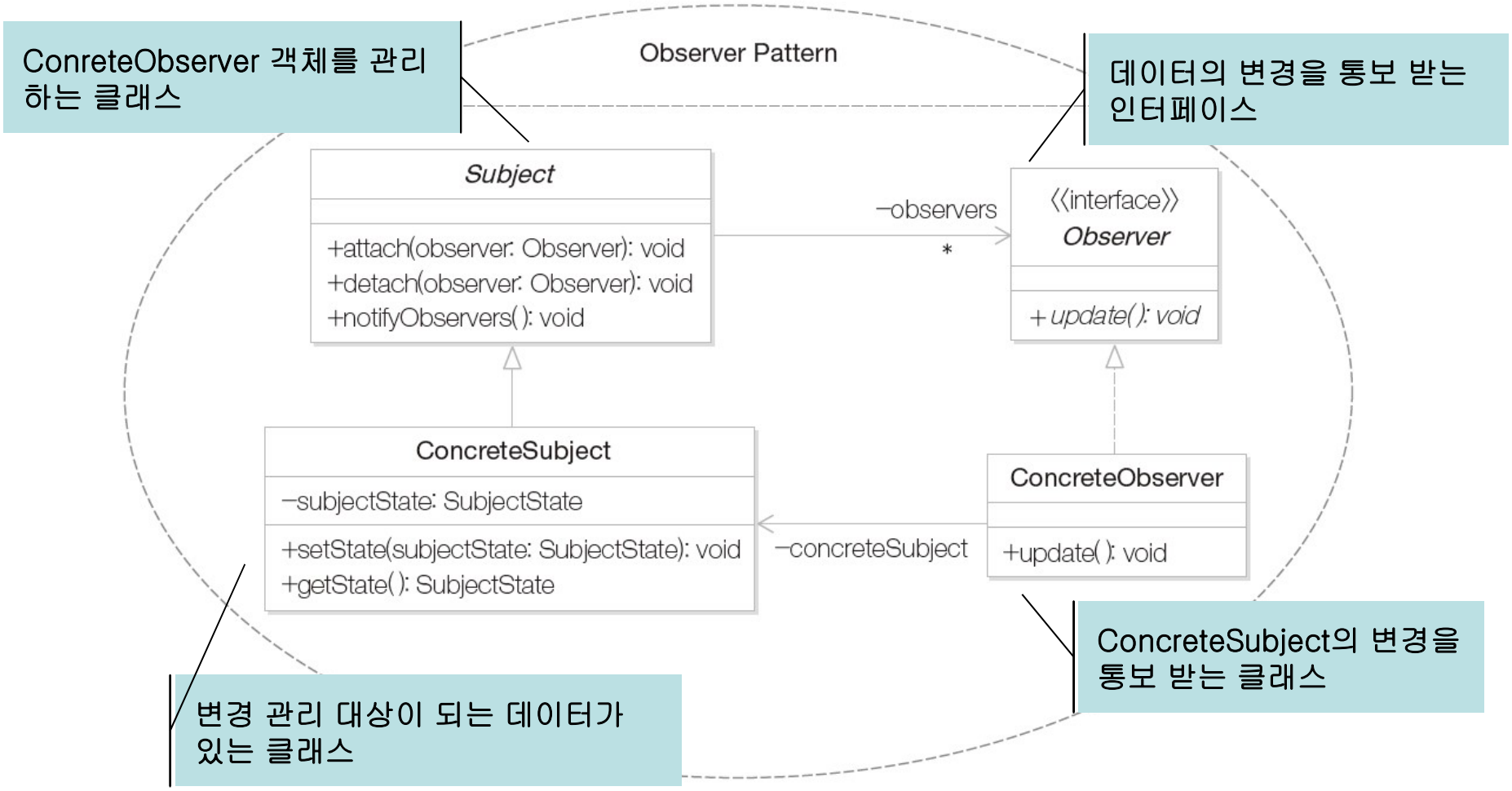
- **데이터의 변경이 발생하였을 때 상대 클래스 및 객체에 의존하지 않으면서 데이터 변경을 통보하고자 할 때**
  - **예) 새로운 파일이 추가되거나 기존 파일이 삭제되었을때 탐색기는 이를 즉시 표시할 필요가 있음**
  - **예) 탐색기를 복수 개 실행하는 상황이나 하나의 탐색기에서 파일 시스템을 변경했을때 다른 탐색기에 즉각적으로 이 변경을 통지해야함**

  - **예) 연료량 클래스와 연료량의 변화에 관심을 갖는 클래스**
    - 차량의 연료가 소진될때까지의 주행가능거리를 출력하는 클래스
    - 연료량이 부족하면 경고메세지를 보내는 클래스
    - 연료량이 부족하면 자동으로 근처 주유소를 표시하는 클래스
    - ➔ 연료량 클래스가 연료량에 관심을 갖는 클래스와 직접 의존하지 않는 방식으로 설계

옵서버 패턴은 통보 대상 객체의 관리를 Subject 클래스와 Observer 인터페이스로 일반화한다. 그러면 데이터 변경을 통보하는 클래스(ConcreteSubject)는 통보 대상 클래스/객체ConcreteObserver)에 대한 의존성을 제거할 수 있다. 결과적으로 옵서버 패턴은 통보 대상 클래스나 대상 객체의 변경에도 ConcreteSubject 클래스를 수정 없이 그대로 사용할 수 있도록 한다.

9.4 옵서버 패턴

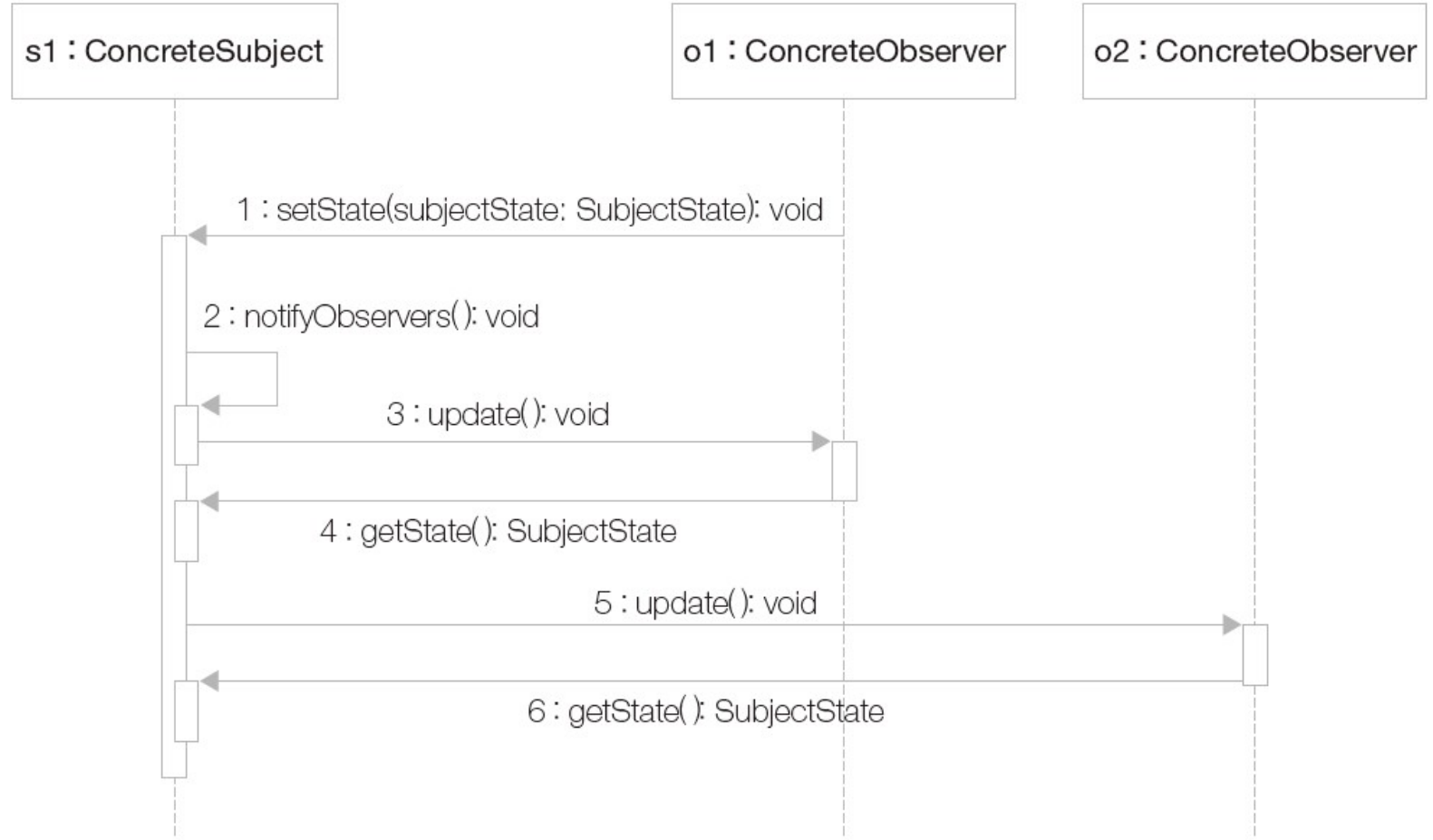
그림 9-4 옵서버 패턴의 컬레보레이션



- **Observer**: 데이터의 변경을 통보 받는 인터페이스. 즉, Subject에서는 Observer 인터페이스의 update 메서드를 호출함으로써 ConcreteSubject의 데이터 변경을 ConcreteObserver에게 통보한다.
- **Subject**: ConreteObserver 객체를 관리하는 요소. Observer 인터페이스를 참조해서 Concrete Observer를 관리하므로 ConcreteObserver의 변화에 독립적일 수 있다.
- **ConcreteSubject**: 변경 관리 대상이 되는 데이터가 있는 클래스. 데이터 변경을 위한 메서드인 setState가 있으며 setState에서는 자신의 데이터인 subjectState를 변경하고 Subject의 notifyObservers 메서드를 호출해서 ConcreteObserver 객체의 변경을 통보한다.
- **ConcreteObserver**: ConcreteSubject의 변경을 통보받는 클래스. Observer 인터페이스의 update 메서드를 구현함으로써 변경을 통보받는다. 변경된 데이터는 ConcreteSubject의 getState 메서드를 호출함으로써 변경을 조회한다.

9.4 옵서버 패턴

그림 9-5 옵서버 패턴의 순차 다이어그램



옵서버 패턴의 적용

그림 9-6 옵서버 패턴을 성적 출력하기 예제에 적용한 경우