Software Architecture

학 기	2020년 2학기
과목명	소프트웨어 아키텍처
교수명	김정호
제출일	2020-10-26
전 공	응용소프트웨어
학 번	60181664
이 름	정 민

Homework Assignment

보고서 및 논문 윤리 서약

1. 나는 보고서 및 논문의 내용을 조작하지 않겠습니다.

2. 나는 다른 사람의 보고서 및 논문의 내용을 내 것처럼 무단으로 복사하지 않겠습니다.

3. 나는 다른 사람의 보고서 및 논문의 내용을 참고하거나 인용할 시 참고 및 인용 형식을 갖추고 출처를 반드시 밝히겠습니다.

4. 나는 보고서 및 논문을 대신하여 작성하도록 청탁하지도 청탁받지도 않겠습니다.

나는 보고서 및 논문 작성 시 위법 행위를 하지 않고, 명지인으로서 또한 공학인으로서 나의 양심과 명예를 지킬 것을 약속합니다.

학 과 : 융합소프트웨어학부
과 목 : 소프트웨어 아키텍처
담당교수 : 김정호 교수님
학번 : 60181664
이름 : 정 민 (서명)

Index

1. 서론

1) Software

Software System이란 하나 이상의 요소들이 모여 하나 이상의 목적을 완전히 달성하는 것이다. Software(이하 SW)는 Stakeholders의 Concerns이 담겨있기 때문에 모호성을 가지고 있다. 그래서 모호성을 없애주는 작업인 분석을 하여 시스템 요구사항을 정리한다.
이렇게 모호성을 없앤 SW는 기능 요구사항과 비기능 요구사항으로 나뉘게 되는데 이때, 비기능 요구사항에 속하는 유지보수성이 매우 중요하다.

SW는 만들어놓으면 일반 부품처럼 고장 나지 않기 때문에 시간이 오래 지나도 계속해서 사용할 수 있다. 하지만 오랜 시간동안 사용하려면 유지보수성이 좋도록 코드를 구현하는 것이 필수적이다. 유지보수성이란 한 번 설계해서 구현한 코드를 얼마나 덜 고치는지에 대한 내용이다.
즉, 구현해놓은 결과물에서 하나를 고쳤을 때 다른 부분도 고치게 되는 것은 유지보수성이 좋지 않다고 할 수 있다.

이렇게 코드를 설계하고 구현하는 데에 있어서 절대로 빠질 수 없는 유지보수성을 공부하기 위해 Pipes & Filters와 Event-Bus에 대해 공부해 보았다.

2) Architecture

Architecture(이하 아키텍처)란 내가 보고자 하는 품질의 구조이다. 그리고 이번에 공부한 Pipes & Filters와 Event-Bus는 실행 시점에 나타나는 SW구조인 Run-Time 구조의 합치기 전략이다.

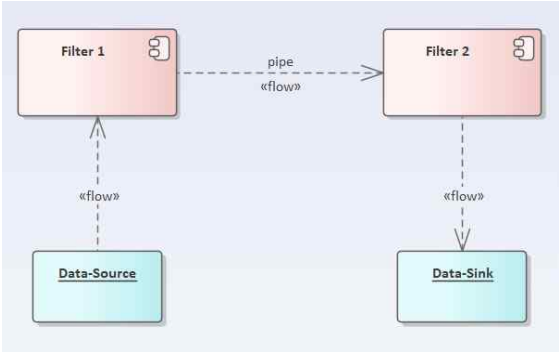
(1) Pipes & Filters

Pipes & Filters는 지속적으로 데이터를 받아서 필터링 결과를 지속적으로 내보내주는 방식이다.

특징은 다음과 같다.
① 실시간성을 유지할 수 있다.
② 변경 여파가 작다.
③ 특정한 영역이 아니면 개발하기 어렵다.

Pipes & Filters가 사용되는 예시는 음향기기와 전투기기 등이 있다.

기본적인 구조는 아래의 그림과 같다.



[그림1] Pipes & Filters 기본 구조

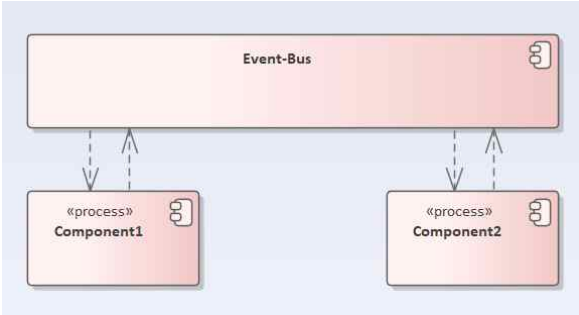
(2) Event-Bus

Event_Bus는 특정한 컴포넌트가 발생한 이벤트를 이벤트 버스가 받아서 등록되어 있는 타 컴포넌트에게 전송하는 방식이며, 두 가지 방식이 있다.
① Implicit(암시적) 방식 : 등록된 모든 컴포넌트에게 전송하는 방식
② Explicit(명시적) 방식 : 특정한 컴포넌트에게만 전송하는 방식

특징은 다음과 같다.
① 컴포넌트 변화 여파가 약하다.
② 안정성이 떨어진다.

Event-Bus를 사용하는 예시로는 CAN Bus와 GUI에서의 이벤트 처리 방식이 있다.

기본 구조는 아래의 그림과 같다.



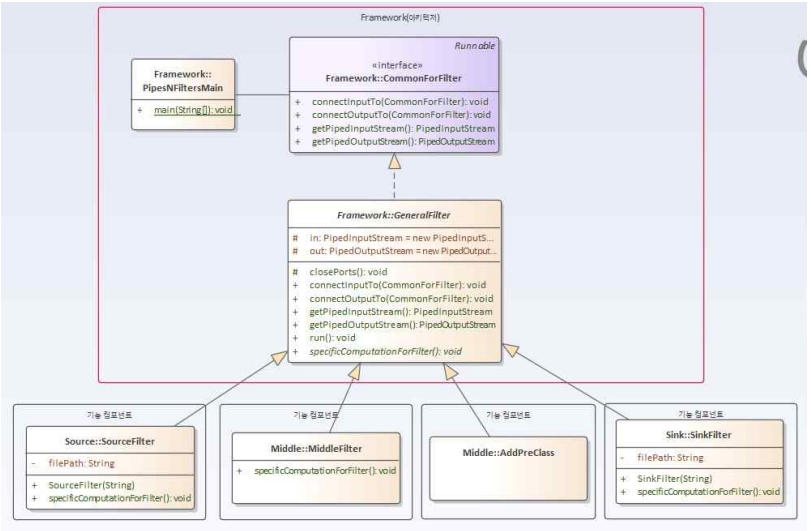
[그림2] Event-Bus 기본 구조

2. 본론

1) Pipes & Filters 문제 A

Q : 모든 CS 학생들은 12345와 23456 과목을 수강해야 한다. 이들 과목을 수강 신청하지 않은 학생들을 골라 이들 과목 ID를 추가하시오.
Q : 모든 CS 학생들은 17651 과목을 수강해야 한다. 이들 과목을 수강 신청하지 않은 학생들을 골라 이들 과목 ID를 추가하시오.

(1) 설계



Pipes & Filters를 이와 같은 방식으로 설계한 이유는 Filter에 상속받아 여러 개의 Filters의 확장과 조립을 편리하게 하기 위함이다. 따라서 Filter Interface를 구현해놓고 이곳에 큰 틀을 구현한 뒤 각각의 Filter의 할 일에 따라 Filters의 내부를 바꾸어주었다.

이로써 위에서(-4-) 설명했듯이 데이터를 지속적으로 받아서 원하는 결과를 받을 수 있게 해주었다.

(2) 구현

Framework :: PipesNFiltersMain.java

```
package Framework;

import Components_Middle.AddPreClass;

public class PipesNFiltersMain {

    public static void main(String[] args) {
        try {

            CommonForFilter filter1 = new SourceFilter("data/Students.txt");
            CommonForFilter filter2 = new SinkFilter("data/Output.txt");
            CommonForFilter filter3 = new MiddleFilter();
            CommonForFilter filter4 = new AddPreClass();

            // sourceFilter와 middleFilter 연결
            filter1.connectOutputTo(filter3);
            // middleFilter와 AddPreClass 연결
            filter3.connectOutputTo(filter4);
            // AddPreClass와 sinkFilter 연결
            filter4.connectOutputTo(filter2);

            // thread 세팅
            Thread thread1 = new Thread(filter1);
            Thread thread2 = new Thread(filter2);
            Thread thread3 = new Thread(filter3);
            Thread thread4 = new Thread(filter4);

            // thread start
            thread1.start();
            thread2.start();
            thread3.start();
            thread4.start();

        } catch(Exception e) {
            e.printStackTrace();
        }
    }

}
```

SourceFilter와 SinkFilter에 각각 Students.txt와 Courses.txt를 연결해주고 SourceFilter - MiddleFilter - AddPreClass - SinkFilter순으로 Filter를 연결해준다.

Framework :: GeneralFilter.java

```java
package Framework;

import java.io.EOFException;

public abstract class GeneralFilter implements CommonForFilter {
    protected PipedInputStream in = new PipedInputStream();
    protected PipedOutputStream out = new PipedOutputStream();

    // 나가는 pipe와 다음 filter의 들어가는 pipe를 연결
    public void connectOutputTo(CommonForFilter nextFilter) throws IOException {
        out.connect(nextFilter.getPipedInputStream());
    }

    // 전 filter에서 나오는 pipe와 들어가는 pipe를 연결
    public void connectInputTo(CommonForFilter previousFilter) throws IOException {
        in.connect(previousFilter.getPipedOutputStream());
    }

    public PipedInputStream getPipedInputStream() {
        return in;
    }

    public PipedOutputStream getPipedOutputStream() {
        return out;
    }

    // Main에서 thread.start()하면 run()실행해서 filter로 보냄
    public void run() {
        try {
            specificComputationForFilter();
        } catch (IOException e) {
            if (e instanceof EOFException)
                return;
            else
                System.out.println(e);
        } finally {
            try {
                out.close();
                in.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
```

GeneralFilter에서 Filter의 큰 틀을 정해준다. output에서 input으로 연결되는 out과 input에서 output으로 연결되는 in통로를 만들어준다. CommonForFilter.java는 GeneralFilter의 Interface Class이다.

Components_Source :: SourceFilter.java

```java
package Components_Source;

import java.io.BufferedInputStream;

public class SourceFilter extends GeneralFilter {
    private String filePath;

    // Students.txt data를 읽어서 filePath에 저장
    public SourceFilter(String inputFilePath) {
        this.filePath = inputFilePath;
    }

    @Override
    public void specificComputationForFilter() throws IOException {
        // txt파일 읽은 것을 out에 써서 다음 filter인 MiddleFilter로 전달
        int byte_read;
        try {
            BufferedInputStream br = new BufferedInputStream(new FileInputStream(new File(filePath)));
            for (;;) {
                byte_read = br.read();
                if (byte_read == -1) {
                    return;
                }
                out.write(byte_read);
            }
        } catch (IOException e) {
            if (e instanceof EOFException)
                return;
            else
                System.out.println(e);
        } finally {
            try {
                out.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}
```

SinkFilter에서는 Student.txt 파일을 읽은 것을 MiddleFilter에 전송해준다. SinkFilter에서 딱히 데이터 정제를 하지 않는다. byte_read == -1, 즉 읽을 내용이 없으면 종료하고 그게 아니라면 읽어서 out pipe에 적어서 전송해준다.

Components_Middle :: MiddleFilter.java

```java
package Components_Middle;

import java.io.IOException;

public class MiddleFilter extends GeneralFilter {

    @Override
    public void specificComputationForFilter() throws IOException {
        // 한 byte를 저장할 인수
        int byte_read = 0;
        // 전공이 나오는 공백 : 4
        int checkBlank = 4;
        // 공백의 개수
        int numOfBlank = 0;
        // SourceFilter에서 읽어온 byte를 저장할 buffer
        byte[] buffer = new byte[64];
        int idx = 0;
        boolean isCS = false;

        while (true) {
            while (byte_read != '\n' && byte_read != -1) {
                // student.txt를 한 줄씩 읽어오고 -1이면 더이상 읽을 내용이 없으므로 종료
                byte_read = in.read();
                if (byte_read == ' ') {
                    // 공백의 개수를 저장
                    numOfBlank++;
                }
                if (byte_read != -1) {
                    // 아직 읽을 것이 남았으면 buffer 크기 늘려서 값 저장
                    buffer[idx++] = (byte) byte_read;
                }
                if (numOfBlank == checkBlank && buffer[idx - 3] == 'C' && buffer[idx - 2] == 'S') {
                    // CS전공인 학생을 구분
                    isCS = true;
                }
            }
```

MiddleFilter에서는 SourceFilter에서 받아온 학생 정보를 정제하는 Filter이다. 문제에 맞게 CS인 학생들을 먼저 선별해주는 작업을 진행한다. Students.txt 파일을 살펴보면 다음과 같은 방식으로 이루어져 있다.

```
20090421 Kim Jason EE 17651 17652 17653 17654
```

데이터 공백의 수가 4일 때, 그로부터 -3글자가 학생의 전공이므로 이를 고려하여 CS전공인 학생들을 선별해준다.

Components_Middle :: MiddleFilter.java (이어서)

```java
            // CS전공인 학생들을 out에 써서 다음 MiddleFilter인 AddPreClass에 전달
            if (isCS == true) {
                for (int i = 0; i < idx; i++) {
                    out.write((char) buffer[i]);
                }
                isCS = false;
            }
            // 다 읽으면 while문 탈출
            if (byte_read == -1) {
                System.out.println("::filtering is finished;");
                return;
            }
            // 변수 초기화
            idx = 0;
            numOfBlank = 0;
            byte_read = '\0';
        }
    }

}
```

CS전공생들이 선별되었으면 이를 다음 MiddleFilter인 AddPreClass에 전송해준다.

Components_Middle :: AddPreClass.java

```java
package Components_Middle;

import java.io.IOException;

public class AddPreClass extends GeneralFilter {

    public void specificComputationForFilter() throws IOException {
        // 한 byte 저장 인수
        int byte_read = 0;
        int idx = 0;
        // MiddleFilter에서 읽어온 byte를 저장할 buffer
        byte[] buffer = new byte[1000];
        // 읽어온 byte중에 학생 정보만 저장하는 stringBuffer
        StringBuffer stu = new StringBuffer();
        // 필수과목(12345, 23456, 17651)을 추가해서 저장하는 stringBuffer
        StringBuffer plus = new StringBuffer();

        while (byte_read != -1) { // course파일 읽어서 stringBuffer에 저장
            byte_read = in.read();
            buffer[idx] = (byte) byte_read;
            stu.append((char) buffer[idx]);
            idx++;
        }
```

MiddleFilter에서 넘어온 CS전공생 데이터를 읽어서 stringBuffer에 저장해준다.

Components_Middle :: AddPreClass.java (이어서)

```
        String stu2 = stu.toString();
        // 줄단위로 문자열 읽기
        String[] stuLine = stu2.split(System.getProperty("line.separator"));

        for (int i = 0; i < stuLine.length; i++) {
            if (!stuLine[i].contains("12345") || !stuLine[i].contains("23456")) {
                if (!stuLine[i].contains("12345") && !stuLine[i].contains("23456")) {
                    // 12345, 23456 모두 듣지 않은 학생일 경우 두 과목 추가
                    plus.append(stuLine[i] + " 12345" + " 23456" + "\n\r");
                } else if (!stuLine[i].contains("12345")) {
                    // 12345를 듣지 않은 학생일 경우 12345 추가
                    plus.append(stuLine[i] + " 12345" + "\n\r");
                } else if (!stuLine[i].contains("23456")) {
                    if (!stuLine[i].contains("17651")) {
                        // 23456와 17651를 듣지 않은 학생일 경우 두 과목 추가
                        plus.append(stuLine[i] + " 23456" + " 17651" + "\n\r");
                    } else {
                        // 17651은 듣고, 23456을 듣지 않은 학생일 경우 23456 추가
                        plus.append(stuLine[i] + " 23456" + "\n\r");
                    }
                }
            } else if (!stuLine[i].contains("17651")) {
                // 17651 를 듣지 않은 학생일 경우
                plus.append(stuLine[i] + " 17651" + "\n\r");
            }
            if (stuLine[i].contains("12345") && stuLine[i].contains("23456") && stuLine[i].contains("17651")) {
                // 모든 필수 과목을 다 들은 학생(추가하지 않아도 되는 학생)일 경우 그대로 가져온다.
                plus.append(stuLine[i] + "\n\r");
            }

        }
        String plus2 = new String();
        plus2 = plus.substring(0, 471) + plus.substring(472, 478);
        String stuFinal = plus2.toString();
        out.write(stuFinal.getBytes());
        // 과목 추가해서 정리한 목록을 string으로 가져와서 정리해준 후 작성
    }

}
```

이후 데이터의 길이만큼 데이터를 확인하며 학생들이 필수로 들어야 하는 과목(12345, 23456, 17651)들을 들었는지 확인하며 만약 듣지 않았을 경우 그 과목을 추가한 후 stringBuffer에 저장해준다.
정제가 끝났으면 String에 저장해준 후 SinkFilter로 전송해준다.

Components_Sink :: SinkFilter.java

```
package Components_Sink;

import java.io.FileWriter;

public class SinkFilter extends GeneralFilter {
    private String filePath;

    // Students.txt data를 읽어서 filePath에 저장
    public SinkFilter(String outputFilePath) {
        this.filePath = outputFilePath;
    }

    @Override
    public void specificComputationForFilter() throws IOException {
        // 한 byte 저장 인수
        int byte_read = 0;

        try {
            // file 생성
            FileWriter fw = new FileWriter(this.filePath);
            for (;;) {
                byte_read = in.read();
                if (byte_read == -1) { // 다 썼으면 종료
                    fw.close();
                    System.out.print("::Filtering is finished; Output file is created.");
                    return;
                } else { // 쓸 게 남았으면 file에 작성
                    fw.write(byte_read);
                }
            }
        } catch (Exception e) {
            closePorts();
            e.printStackTrace();

        }
    }

}
```

AddPreClass(MiddleFilter)에서 정제된 데이터를 받아온 뒤, file에 작성해준다. PipesNFiltersMain에 나와있듯이 SinkFilter는 Output.txt와 연결되어 있으므로 Output.txt에 작성된다. 다 작성하였으면 종료한다.

(3) 결과

코드 구현 후 실행한 결과는 다음과 같다.

```
20100123 Hwang Myunghan CS 12345 23456 17651 17652
20110512 Park Hongsun CS 12345 23456 17652 17654 17651
20100123 Kim Yunmi CS 12345 17651 17652 17653 23456
20100323 Jung Philsoo CS 12345 23456 17651 17655
20070452 Ko Kyungmin CS 12345 17652 17655 23456 17651
20130091 Kim Chulmin CS 12345 23456 17651 17652
20100128 Kim Minsu CS 12345 17651 17652 17653 23456
20100131 Kim JungMi CS 17651 17652 17653 17655 12345 23456
20130094 Jang Goyoung CS 12345 23456 17653 17651
```

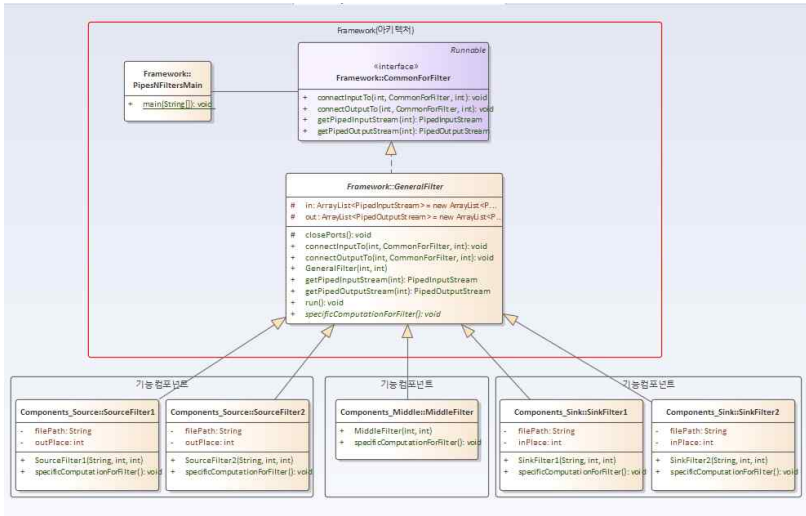
(4) 느낀점

GeneralFilter에 Filter의 큰 틀을 만들어두고 Source, Middle, Sink 의 내부만 조금씩 바꾸는 방식은 Filter를 추가하거나 분리하기에 편리 하였다. 원래 이차원 배열을 사용하여 코드를 구현했었다. 그때는 추가 되는 과목이 학생 데이터 뒤에 이어서 출력되지 않고 다음 줄로 넘어가서 출력이 되었는데, stringBuffer를 이용하여 그 점을 수정하였다.
하지만 Output.txt을 출력하면 몇 개의 문자와 출력본 중간에 띄어쓰기가 한 번 더 되는 현상이 발생했고, 아직 해결법을 찾지 못하였다.
지금은 substring()으로 해결을 하였지만, 이것의 해결법에 대해 알아보고 싶다.

2) pipes & Filters 문제 B

Q : 학생 데이터와 과목 데이터를 읽고 선수과목을 잘 수강한 학생은 Output1.txt로, 선수과목을 다 수강하지 않은 학생은 Output2.txt로 각각 출력하시오.

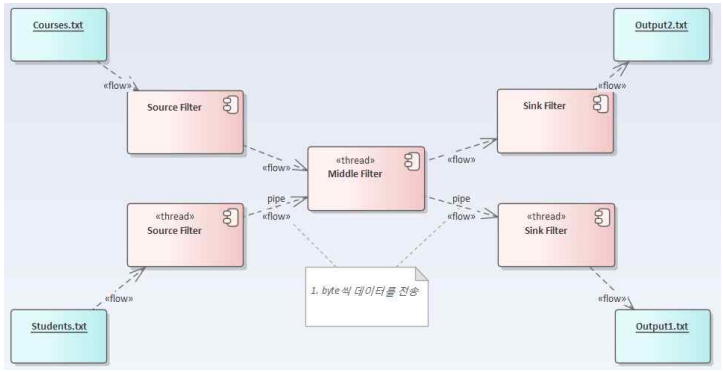
(1) 설계



Pipes & Filters 문제 B에 관련하여 이와 같은 방식으로 설계한 방식은 A와 마찬가지로 여러 개의 Filters의 확장과 조립을 편리하게 하기 위함 이다. 여기서 더 변화한 점은 문제 A와는 다르게 Students.txt와 Courses.txt 두 개의 파일을 모두 읽어와야 했고, 또 두 가지의 Output.txt 파일로 출력해줘야하기 때문에 파일을 읽는 SourceFilter와 파일을 작성하여 출력하는 SinkFilter를 두 개씩 만들어주었다.

.

문제 B의 Runtime View는 다음과 같다.



(2) 구현

Framework :: PipesNFiltersMain.java

```
import Components_Middle.MiddleFilter;

public class PipesNFiltersMain {

    public static void main(String[] args) {
        try {

            CommonForFilter filter1 = new SourceFilter1("Students.txt", 1, 1); // 학생정보 읽은 것을 input필터 첫번째 빈지에 넣어줌
            CommonForFilter filter2 = new SourceFilter2("Courses.txt", 1, 2); // 학생정보 읽은 것을 input필터 두번째 빈지에 넣어줌

            CommonForFilter filter3 = new MiddleFilter(2, 2); // sourceFilter와 SinkFilter를 연결해주는데 2개의 빈지를 가짐

            CommonForFilter filter4 = new SinkFilter1("Output1.txt", 1, 1); // filtering끝난 결과를 Output1.txt에 출력
            CommonForFilter filter5 = new SinkFilter2("Output2.txt", 1, 1); // filtering끝난 결과를 Output2.txt에 출력

            filter1.connectOutputTo(0, filter3, 0); // SourceFilter1과 MiddleFilter 0번지 연결
            filter2.connectOutputTo(1, filter3, 1); // SourceFilter2와 MiddleFilter 1번지 연결

            filter3.connectOutputTo(0, filter4, 0); // MiddleFilter 0번지와 SinkFilter연결
            filter3.connectOutputTo(1, filter5, 0); // MiddleFilter 1번지와 SinkFilter연결

            // thread 세팅
            Thread thread1 = new Thread(filter1);
            Thread thread2 = new Thread(filter2);

            Thread thread3 = new Thread(filter3);

            Thread thread4 = new Thread(filter4);
            Thread thread5 = new Thread(filter5);

            // thread start
            thread1.start();
            thread2.start();
            thread3.start();
            thread4.start();
            thread5.start();

        } catch (Exception e) {
            e.printStackTrace();
        }
    }

}
```

SourceFilter 2개, SinkFilter 2개가 필요하기에 2개씩 만들어준 다음에 index를 지정해주었다.
SourceFilter는 out pipe를 통해서 MiddleFilter로 데이터를 넘겨줘야 하기 때문에 (out pipe 하나, index)의 의미로 (1, 1), (1, 2)로 설정을 해주었고, MiddleFilter는 Source Filter에서 두 가지 정보를 받고, SinkFilter로 두 가지 정보를 보내주기 때문에 (2, 2)로 설정을 해주었다.
마지막으로 SinkFilter는 MiddleFilter에서 각각 따로 정보를 받기 때문에 (int pipe 하나, index)의 의미로 (1, 1)로 설정해주었다.
그리고 각각의 index에 맞게 Filter를 연결해주었다.

Framework :: GeneralFilter.java

```java
package Framework;

import java.io.EOFException;

public abstract class GeneralFilter implements CommonForFilter {
    // 한 byte씩 전달
    protected ArrayList<PipedInputStream> in = new ArrayList<PipedInputStream>();
    protected ArrayList<PipedOutputStream> out = new ArrayList<PipedOutputStream>();

     // input필터와 output필터에 2개의 빈공간 생성
    public GeneralFilter(int inPlace, int outPlace) {
        for(int i = 0; i < inPlace; i++) {
            in.add(new PipedInputStream());
        }
        for(int i = 0; i < outPlace; i++) {
            out.add(new PipedOutputStream());
        }
    }

    // 나가는 pipe와 다음 filter의 들어가는 pipe를 연결
    public void connectOutputTo(int outputIndex, CommonForFilter previousFilter, int inputIndex) throws IOException {
        out.get(outputIndex).connect(previousFilter.getPipedInputStream(inputIndex));
    }

    // 전 filter에서 나오는 pipe와 들어가는 pipe를 연결
    public void connectInputTo(int inputIndex, CommonForFilter previousFilter, int outputIndex) throws IOException {
        in.get(inputIndex).connect(previousFilter.getPipedOutputStream(outputIndex));
    }

    public PipedInputStream getPipedInputStream(int index) {
        return in.get(index);
    }

    public PipedOutputStream getPipedOutputStream(int index) {
        return out.get(index);
    }
```

문제 B에서는 SourceFilter와 SinkFilter가 2개씩 필요하고, 각각의 Filter에서 데이터를 읽어오기 때문에 pipe를 ArrayList로 만들어주었다. 그리고 각각의 Filters에서 정보가 잘 저장되고, 전달될 수 있도록 빈공간 2개씩을 각각 in과 out에 생성해주었다.
CommonForFilter.java는 GeneralFilter의 Interface Class이다.

Components_Middle :: MiddleFilter.java

```java
package Components_Middle;
import java.io.IOException;

public class MiddleFilter extends GeneralFilter {

    public MiddleFilter(int inPlace, int outPlace) {
        super(inPlace, outPlace);
    }

    @Override
    public void specificComputationForFilter() throws IOException {
        // 한 byte씩 저장할 인수
        int byte_read = 0;
        // 변지
        int i = 0;
        int idx = 0;
        // 읽은 파일을 저장할 buffer
        byte[] buffer = new byte[2056];
        // Students.txt만 저장하는 stringBuffer
        StringBuffer stu = new StringBuffer();
        // Courses.txt만 저장하는 stringBuffer
        StringBuffer cour = new StringBuffer();
        // Students.txt, Courses.txt 모두 저장하는 stringBuffer
        StringBuffer all = new StringBuffer();
        // 선수과목을 잘 들은 학생들을 저장하는 stringBuffer
        StringBuffer stuOk = new StringBuffer();
        // 선수과목을 잘 듣지 않은 학생들을 저장하는 stringBuffer
        StringBuffer stuNo = new StringBuffer();

        while (true) {
            while (i < 2) { // 파일 전달 받아서 읽은 후에 stringBuffer에 저장
                byte_read = in.get(i).read();
                if (byte_read != -1) {
                    buffer[idx] = (byte) byte_read;
                    all.append((char) buffer[idx]);
                    idx++;
                }

                if (byte_read == -1) {

                    if (i == 0) { // Students.txt를 읽고 stringBuffer에 저장
                        stu.append(all);
                        System.out.println("::Student filtering is finished;");
                    }
                    if (i == 1) { // Students.txt, Courses.txt를 읽고 stringBuffer에 저장
                        cour.append(all);
                        cour.delete(0, 910); // Students.txt부분 삭제 > Courses.txt부분만 저장
                        System.out.println("::Course filtering is finished");
                    }
                    i++;
                }
            }
```

Students.txt와 Courses.txt를 각각 정제해서 사용하기 위해서 같이 읽어온 두 개의 데이터를 각각 분리해서 저장해준다.

Components_Middle :: MiddleFilter.java (이어서)

```
String cour2 = cour.toString();
// 줄단위로 문자열 읽기
String[] courLine = cour2.split(System.getProperty("line.separator"));

String stu2 = stu.toString();
String[] stuLine = stu2.split(System.getProperty("line.separator"));

String c1 = "";
String c2 = "";
String c3 = "";
String c4 = "";

// 17651의 선수과목 12345 저장
for (int k = 0; k < courLine.length; k++) {
    if (courLine[k].contains("17651") && !courLine[k].contains("17655") && !courLine[k].contains("17652")
            && !courLine[k].contains("17654")) {
        c1 = courLine[k];
        c1 = c1.substring(c1.length() - 6, c1.length() - 1);
    }
}

// 17652의 선수과목 23456 저장
for (int k = 1; k < courLine.length; k++) {
    if (courLine[k].contains("17652") && !courLine[k].contains("17655") && !courLine[k].contains("17651")
            && !courLine[k].contains("17654")) {
        c2 = courLine[k];
        c2 = c2.substring(c2.length() - 6, c2.length() - 1);
    }
}

// 17655의 선수과목 12345, 17651 저장
for (int k = 0; k < courLine.length; k++) {
    if (courLine[k].contains("17655") && courLine[k].contains("17651") && !courLine[k].contains("17652")
            && !courLine[k].contains("17654")) {
        c3 = courLine[k];
        c3 = c3.substring(c3.length() - 12, c3.length());
        c4 = c3.substring(c3.length() - 11, c3.length() - 6); // 12345
        c3 = c3.substring(c3.length() - 5, c3.length()); // 17651

    }
}
```

먼저 Courses.txt부터 정제해주었다. 선수과목에 따라 학생들을 구별해야 하기 때문에 선수과목을 따로 뽑아준 후 각각 String에 저장해준다.

Components_Middle :: MiddleFilter.java (이어서)

```
for (int l = 0; l < stuLine.length; l++) {
    if (stuLine[l].contains("17655")) {
        if (stuLine[l].contains(c4) && stuLine[l].contains(c3)) { // 17655 12345 17651 들은 학생일 경우
            stuOk.append(stuLine[l] + "\n\r"); // stuOk : 선수과목을 모두 잘 들은 학생 저장
        } else { // 17655는 들었으나 선수과목을 하나라도 안 들은 학생일 경우
            stuNo.append(stuLine[l] + "\n\r"); // stuNo : 선수과목을 모두 듣지 않은 학생 저장
        }
    } else if (stuLine[l].contains("17652")) {
        if (stuLine[l].contains(c2)) { // 17652 23456 들은 학생일 경우
            stuOk.append(stuLine[l] + "\n\r");
        } else { // 선수과목 23456 안 들은 학생일 경우
            stuNo.append(stuLine[l] + "\n\r");
        }
    } else if (stuLine[l].contains("17654") && stuLine[l].contains(c3)) { // 17654 17651 들은 학생
        if (stuLine[l].contains("12345")) { // 17654 17651 12345 들은 학생일 경우
            stuOk.append(stuLine[l] + "\n\r");
        } else { // 17654 17651만 들은 학생일 경우(선수과목 12345를 안 들은 경우)
            stuNo.append(stuLine[l] + "\n\r");
        }
    } else if (stuLine[l].contains("17651")) {
        if (stuLine[l].contains(c1)) { // 17651 12345 들은 학생일 경우
            stuOk.append(stuLine[l] + "\n\r");
        } else { // 선수과목 12345 안 들은 학생일 경우
            stuNo.append(stuLine[l] + "\n\r");
        }
    } else if (stuLine[l].contains("12345") && stuLine[l].contains("23456") // 선수과목없는 과목만 들은 학생일 경우
            && stuLine[l].contains("17653")) {
        if (!stuLine[l].contains("17655") && !stuLine[l].contains("17654") && !stuLine[l].contains("17652")
                && !stuLine[l].contains("17651")) {
            stuOk.append(stuLine[l] + "\n\r");
        }
    }
}
String finalNo = stuNo.toString();
out.get(1).write(finalNo.getBytes());
String finalOk = stuOk.toString();
out.get(0).write(finalOk.getBytes());
// 각각 저장한 파일 SinkFilter로 보내기

if (i == 2) {
    i = 0;
    return;
}
```

정제된 Courses.txt와 Students.txt를 활용하여 선수과목을 들은 학생들과 그렇지 않은 학생들을 선별하여 stringBuffer에 각각 저장해준다.
그리고 stringBuffer에 있는 데이터를 다시 String에 저장해준 뒤,
각각 0번지와 1번지에 넣어 SinkFilter로 전송해준다.
그리고 0, 1번지까지밖에 없으므로 i == 2가 되면 종료한다.

Components_Sink :: SinkFilter1.java

```java
package Components_Sink;

import java.io.FileWriter;

public class SinkFilter1 extends GeneralFilter {
    // MiddleFilter로부터 받은 파일 저장할 인수
    private String filePath;
    private int inPlace;

    public SinkFilter1(String outputFilePath, int in, int out) {
        super(in, out);
        this.inPlace = out - 1;
        this.filePath = outputFilePath;
    }

    @Override
    public void specificComputationForFilter() throws IOException {
        // 한byte씩 저장할 인수
        int byte_read = 0;

        try {
            FileWriter fw = new FileWriter(this.filePath);
            for (;;) {
                // MiddleFilter에서 데이터 받아온 거 읽음
                byte_read = in.get(inPlace).read();
                if (byte_read == -1) {
                    fw.close();

                    System.out.print("::Filtering is finished; Output file is created.");
                    return;
                } else {
                    fw.write(byte_read); // 파일 받아서 Output1.txt에 작성

                }
            }
        } catch (Exception e) {
            closePorts();
            e.printStackTrace();

        }
    }

}
```

SinkFilter1에서는 선수과목을 모두 잘 들은 학생의 데이터를 가져온 뒤, Output1.txt에 작성하는 역할을 한다.

Components_Sink :: SinkFilter2.java

```java
package Components_Sink;

import java.io.FileWriter;

public class SinkFilter2 extends GeneralFilter {
    // MiddleFilter로부터 받은 파일 저장할 인수
    private String filePath;
    // MiddleFilter에서 읽어올 데이터의 번지수를 저장할 인수
    private int inPlace;

    public SinkFilter2(String outputFilePath, int in, int out) {
        super(in, out);
        this.inPlace = out - 1;
        this.filePath = outputFilePath;
    }

    @Override
    public void specificComputationForFilter() throws IOException {
        // 한byte씩 저장할 인수
        int byte_read = 0;

        try {
            FileWriter fw = new FileWriter(this.filePath);
            for (;;) {
                // MiddleFilter에서 데이터 받아온 거 읽음
                byte_read = in.get(inPlace).read();
                if (byte_read == -1) {
                    fw.close();
                    System.out.print("::Filtering is finished; Output file is created.");
                    return;
                } else {
                    fw.write(byte_read); // 파일 받아서 Output2.txt에 작성

                }
            }
        } catch (Exception e) {
            closePorts();
            e.printStackTrace();

        }
    }

}
```

SinkFilter2에서는 선수과목을 모두 다 듣지 않은 학생의 데이터를 가져온 뒤, Output2.txt에 작성하는 역할을 한다.

(3) 결과

코드 구현 후 결과는 다음과 같다.

Output1.txt

```
20100123 Hwang Myunghan CS 12345 23456 17651 17652

20110512 Park Hongsun CS 12345 23456 17652 17654

20100125 Kim Hokyung ME 12345 23456 17651 17654

20100323 Jung Philsoo CS 12345 23456 17651 17655

20110298 Lee Mijung ME 12345 23456 17651 17652

20120808 Kim Sungsuk EE 12345 23456 17651 17652

20130091 Kim Chulmin CS 12345 23456 17651 17652

20130094 Jang Goyoung CS 12345 23456 17653

20130095 Kim Soyoung EE 12345 23456 17651 17652
```

- Output1.txt : 선수과목을 잘 들은 학생들

Output2.txt

```
20090421 Kim Jason EE 17651 17652 17653 17654

20100123 Kim Yunmi CS 12345 17651 17652 17653

20080678 Ahn Jonghyuk EE 12345 17651 17652 17654

20080603 Park Kitea ME 17651 17652 17653

20070452 Ko Kyungmin CS 12345 17652 17655

20110876 Park Kiyoung EE 12345 17651 17652 17654

20100128 Kim Minsu CS 12345 17651 17652 17653

20100131 Kim JungMi CS 17651 17652 17653 17655
```

- Output2.txt : 선수과목을 다 듣지 않은 학생들

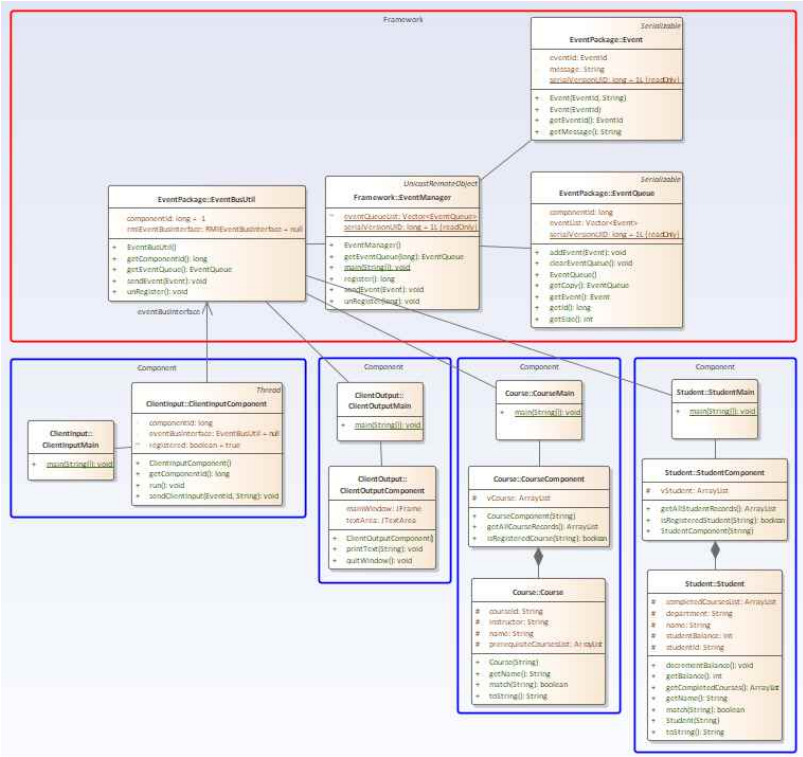
(4) 느낀점

Pipes & Filters 문제 B는 문제 A보다 어려웠고, 그만큼 코드 구현하는 시간도 오래 걸렸다. 각각의 통로를 정해주고 그 길로 데이터가 전송되도록 해야 한다고 생각하는 것이 나에게는 어려웠던 것 같다. 그리고 MiddleFilter에서 데이터를 정제하는 것에도 꽤나 많은 시간을 투자하였다. 평소에 간단하다고 생각했던 if문이었으나 이번에는 두 가지 데이터를 모두 활용해야 했기 때문에 좀 더 어려웠었다. 처음에는 과목 데이터를 생각하지 못하고 학생 데이터로만 데이터를 정제하는 실수도 했었다. 이번 코드를 통해 데이터를 전달하는 통로(pipe)에 대해 더 공부할 수 있었고, 무사히 구현을 완성할 수 있었다.

3) Event-Bus 문제 A

- 요구사항 : 학생 및 과목 정보 삭제 기능 추가

(1) 설계



Event-Bus는 각각의 이벤트들이 Event-Bus에서 이동하기 때문에 학생, 과목 정보를 삭제할 때도 Event-Bus를 이동하도록 설계하였다.
학생 정보 삭제는 StudentMain.java에서 과목 정보 삭제는 CourseMain.java 에서 수행하도록 설계하였다.

(2) 구현

Components.ClientInput :: ClientInputMain.java

```
} else if (selection.contentEquals("5")) {  // 학생 정보 삭제
    String msg = "";
    boolean inputIsDone = false;
    while (!inputIsDone) { // 삭제할 학생 ID 입력
        System.out.println("\nEnter student ID and press return (Ex. 20131234)>> ");
        String sSID = br.readLine().trim();
        msg = sSID;

        while (true) {
            System.out.println("\nIs it correct information? (Y/N)");
            System.out.println(msg);
            String ans = br.readLine().trim();
            if (ans.equalsIgnoreCase("y")) {
                inputIsDone = true;
                break;
            } else if (ans.equalsIgnoreCase("n")) {
                System.out.println("\nType again...");
                msg = "";
                break;
            } else {
                System.out.println("\nTyped wrong answer");
            }
        }
    }

    System.out.println("\n ** Sending an event(ID:" + EventId.DeleteStudents + ") with ");
    System.out.println("\n ** Message \"" + msg + "\" ...");
    clientInputComponent.sendClientInput(EventId.DeleteStudents, msg); // DeleteStudents로 msg(삭제할 학생 ID)와 이벤트 보냄
```

먼저 학생을 삭제하는 기능을 위해 ClientInputMain의 else if문에 추가해주었다. 그리고 학생ID를 입력하면 DeleteStudents로 학생 ID와 함께 이벤트가 넘어간다.

```
} else if (selection.contentEquals("6")) { // 과목 정보삭제
    String userInput = "";
    boolean inputIsDone = false;

    while (!inputIsDone) { // 삭제할 과목 ID 입력
        System.out.println("\nEnter course ID and press return (Ex. 12345)>> ");
        userInput = br.readLine().trim();

        while (true) {
            System.out.println("\nIs it correct information? (Y/N)");
            System.out.println(userInput);
            String ans = br.readLine().trim();
            if (ans.equalsIgnoreCase("y")) {
                inputIsDone = true;
                break;
            } else if (ans.equalsIgnoreCase("n")) {
                System.out.println("\nType again...");
                userInput = "";
                break;
            } else {
                System.out.println("\nTyped wrong answer");
            }
        }
    }

    System.out.println("\nSending an event(ID:" + EventId.DeleteCourses + ") with");
    System.out.println("\n ** Message \"" + userInput + "\" ...");
    clientInputComponent.sendClientInput(EventId.DeleteCourses, userInput); // DeleteCourses로 userInput(삭제할 과목ID)와 이벤트 보냄
```

과목 정보 삭제하는 기능도 마찬가지로 추가해주고 삭제하고자 하는 과목 ID를 입력하면 DeleteCourses로 과목 ID와 함께 이벤트가 넘어간다.

Components.Student :: StudentMain.java

```
} else if (event.getEventId() == EventId.RegisterStudents) {
    String studentInfo = event.getMessage();
    System.out.println("Not null");
    Student student = new Student(studentInfo);
    if (studentsList.isRegisteredStudent(student.studentId) == false) {
        studentsList.vStudent.add(new Student(studentInfo));
        System.out.println("A new student is successfully added...");
        System.out.println("\"" + studentInfo + "\"");
    } else {
        System.out.println("\n ** Sending an event(ID:" + EventId.ClientOutput + ") with");
        System.out.println("\n ** Message: This student is already registered.  ...");
        eventBusInterface
                .sendEvent(new Event(EventId.ClientOutput, "This student is already registered."));
    }
```

DeleteStudents로 이벤트가 넘어오면 StudentMain클래스에 도착한다. 삭제할 학생 ID가 학생 리스트에 존재하면 해당 학생의 index를 찾아 삭제하고, 이벤트 정보를 ClientOutput에 보내고 초기화면으로 돌아간다. 해당 학생의 index를 찾는 것은 StudentComponent.java 에 구현하였다.

Components.Student :: StudentComponent.java

```
public int getStudentIndex(String sSID) {
    for (int i = 0; i < this.vStudent.size(); i++) {
        Student objStudent = (Student) this.vStudent.get(i);
        if (objStudent.match(sSID)) {
            return i;
        }
    }
    return -1;
}
}
```

이곳에서 해당 학생의 index를 찾을 수 있는 함수를 만들어주고, 이를 활용하여 학생 정보를 삭제하였다.

Components.Course :: CourseMain.java

```
} else if (event.getEventId() == EventId.DeleteCourses) { // 과목 정보 삭제
    String courseInfo = event.getMessage();
    if (coursesList.isRegisteredCourse(courseInfo) == true) { // 삭제하고자 하는 과목ID가 과목리스트에 있으면
        coursesList.vCourse.remove(coursesList.getCourseIndex(courseInfo)); // 해당 과목 인덱스 가져와서 삭제
        System.out.println("A course is successfully deleted...");
        System.out.println("\"" + courseInfo + "\"");
    } else {
        System.out.println("\n ** Sending an event(ID:" + EventId.ClientOutput + ") with");
        System.out.println("\n ** Message: This course is already deleted.  ...");
        eventBusInterface.sendEvent(
                new Event(EventId.ClientOutput, "This course is already deleted."));
    }
```

과목 정보 삭제는 학생 정보 삭제와 마찬가지로 구현해주었다. 삭제할 과목 ID가 과목 리스트에 있으면 그 해당 index의 과목 정보를 찾아 삭제한 후 ClientOutput에 정보 전달 후 초기화면으로 돌아간다. 해당 과목의 index를 찾는 것은 CourseComponent.java에서 구현하였다.

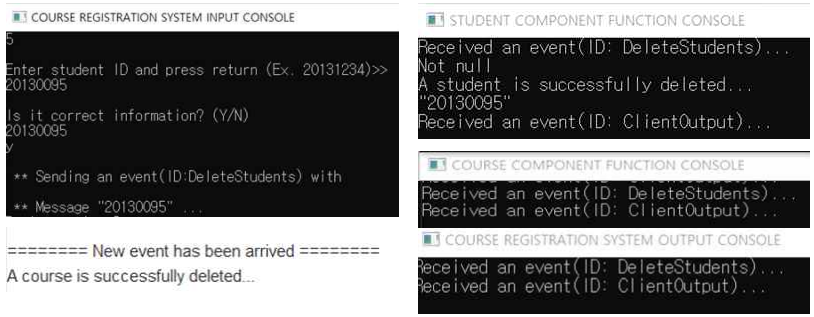
Components.Course :: CourseComponent.java

```
public int getCourseIndex(String courseId) {
    for (int i = 0; i < this.vCourse.size(); i++) {
        Course course = (Course) this.vCourse.get(i);
        if(course.match(courseId)) {
            return i;
        }
    }
    return -1;
}
```

StudentComponent.java와 마찬가지로 해당 과목의 index를 찾도록 하는 함수를 만들어서 CourseMain.java에서 사용할 수 있도록 하였다.

(3) 결과

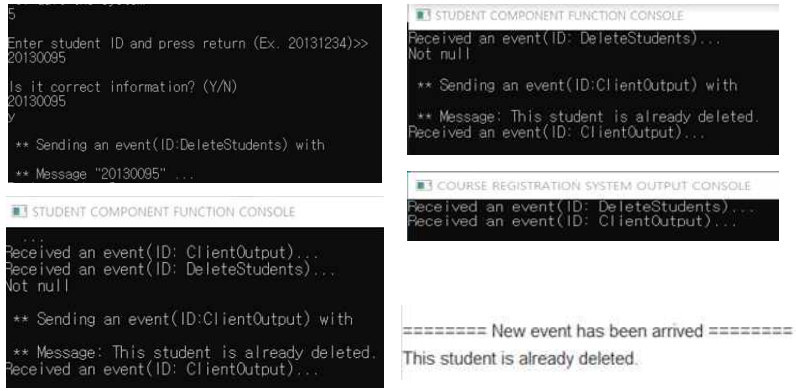
<학생 정보 삭제 결과>



<학생 정보 삭제 후 학생 리스트>

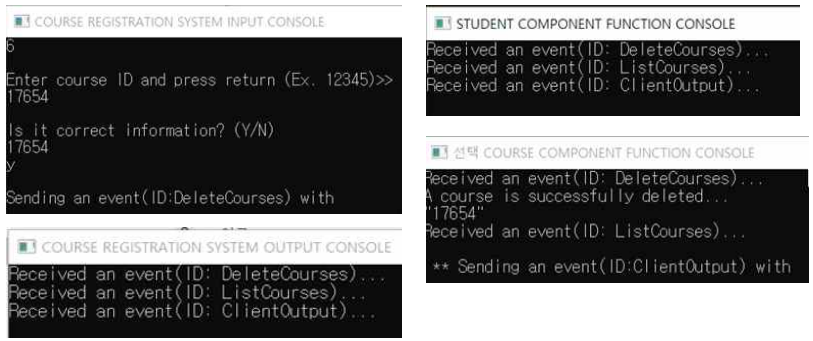
- 20130095 학생이 삭제됨

<존재하지 않는(이미 삭제된) 학생을 삭제할 경우>
- 이미 삭제한 20130095 다시 삭제 시도



- 이미 삭제된 학생이라고 메시지 출력

<과목 정보 삭제 결과>



<과목 정보 삭제 후 과목 리스트>

- 17654 과목이 삭제됨

<존재하지 않는(이미 삭제된) 과목을 삭제할 경우>
- 이미 삭제한 17654 다시 삭제 시도

```
6

Enter course ID and press return (Ex. 12345)>>
17654

Is it correct information? (Y/N)
17654
y

Sending an event(ID:DeleteCourses) with

 ** Message "17654" ...
```

STUDENT COMPONENT FUNCTION CONSOLE

```
Received an event(ID: DeleteCourses)...
Received an event(ID: ClientOutput)...
```

COURSE REGISTRATION SYSTEM OUTPUT CONSOLE

```
Received an event(ID: DeleteCourses)...
Received an event(ID: ClientOutput)...
```

COURSE COMPONENT FUNCTION CONSOLE

```
Received an event(ID: DeleteCourses)...

 ** Sending an event(ID:ClientOutput) with

 ** Message: This course is already deleted.
Received an event(ID: ClientOutput)...
```

```
======== New event has been arrived ========
This course is already deleted.
```

- 이미 삭제된 과목이라고 메시지 출력

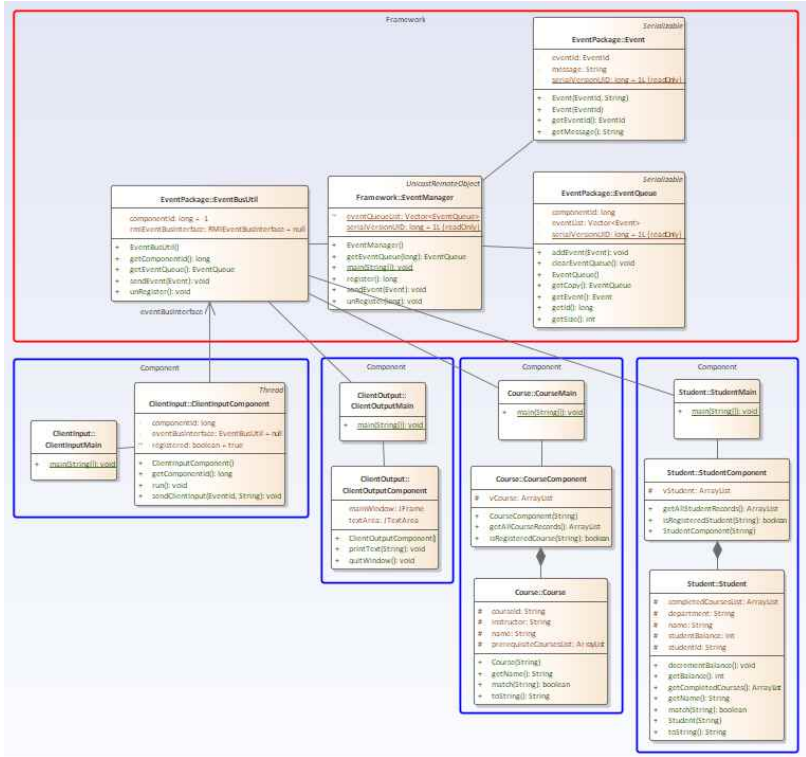
(4) 느낀점

학생 정보와 과목 정보를 삭제하는 기능을 추가하는 것은 그렇게 어렵지 않았다. 먼저 ID를 입력하면 해당 ID에 해당하는 index를 받아와서 그 index에 해당하는 정보를 삭제하도록 구현하였다.
하지만 노트북 성능이 좋지 않아 bat파일을 실행하면 컴퓨터가 매우 느려지는 것이 많이 불편했다.

4) Event-Bus 문제 B

- 요구사항 : 수강 신청 기능 추가

(1) 설계



기본적인 구조는 Event-Bus 문제 A와 동일하다. 모든 이벤트들은 Event-Bus를 통해서 이동한다.
수강 신청은 학생 관련된 정제는 StudentMain.java 에서 하도록 설계하였고, 과목과 관련된 정제는 CourseMain.java에서 하도록 설계하였다.
그리고 Pipes & Filters 문제 B에서 했던 것처럼 학생의 선수과목 수강 여부를 판단하여 수강 신청을 완료하도록 설계하였다.

(2) 구현

Components.ClientInput :: ClientInputMain.java

```
} else if (selection.equals("7")) { // 수강신청
    String msg = "";
    String userInput = "";
    boolean inputIsDone = false;

    while (!inputIsDone) {
        System.out.println("\nEnter student ID and press return (Ex. 20131234)>> ");
        String sSID = br.readLine().trim();
        msg = sSID;

        while (true) {
            System.out.println("\nIs it correct information? (Y/N)");
            System.out.println(msg);
            String ans = br.readLine().trim();
            if (ans.equalsIgnoreCase("y")) {
                // 입력한 학생 ID가 리스트에 있는 학생이면, 과목번호 입력 / n을 누르면 다시 입력
                System.out.println("\nEnter course ID and press return (Ex. 12345)>> ");
                userInput = br.readLine().trim();
                System.out.println("\nIs it correct information? (Y/N)");
                System.out.println(userInput);
                ans = br.readLine().trim();
                if (ans.equalsIgnoreCase("y")) {
                    inputIsDone = true;
                    break;
                } else if (ans.equalsIgnoreCase("n")) {
                    System.out.println("\nType again...");
                    userInput = "";
                    break;
                } else {
                    System.out.println("\nTyped wrong answer");
                }

            } else if (ans.equalsIgnoreCase("n")) {
                System.out.println("\nType again...");
                msg = "";
                break;
            } else {
                System.out.println("\nTyped wrong answer");
            }
        }
    }
        String all =  msg + " " + userInput;
        System.out.println("\nSending an event(ID:" + EventId.Login + ") with");
        System.out.println("\n ** Message \"" + userInput + "\" ...");
        clientInputComponent.sendClientInput(EventId.Login, all); // 학생ID + 과목ID 전송
```

수강 신청 기능을 추가하기 위해 ClientInputMain.java에 수강 신청 항목을 추가하고, 학생 ID와 과목 ID를 입력하면 StudentMain.java에 이벤트와 정보가 넘어간다.

Components.Student :: StudentMain.java

```
} else if (event.getEventId() == EventId.Login) {
    String studentInfo = event.getMessage(); // 학생 + 과목
    if (studentsList.isRegisteredStudent(studentInfo.substring(0, 8)) == true) { // 학생정보 있으면 로그인 성공
        System.out.println("A student is successfully login...");
        System.out.println("\"" + studentInfo + "\"");
        String student = studentsList.readStudent(studentInfo.substring(0, 8)); // 학생 정보
        String courseInfo = studentInfo.substring(studentInfo.length() - 5, studentInfo.length()); // 과목ID

        if (!student.matches(studentInfo.substring(9))) { // 학생이 이 과목을 안 들었으면
            System.out.println("\n ** Message: loading.  ...");
            eventBusInterface.sendEvent(new Event(EventId.Registeration, studentInfo)); // 학생 + 과목

        } else { // 들은 과목이면 초기화면으로
            System.out.println("\n ** Sending an event(ID:" + EventId.ClientOutput + ") with");
            System.out.println("\n ** Message: This student has already taken this course.  ...");
            eventBusInterface.sendEvent(
                    new Event(EventId.ClientOutput, "This student has already taken this course."));
        }

    } else { // 학생 정보가 존재하지 않으면
        System.out.println("\n ** Sending an event(ID:" + EventId.ClientOutput + ") with");
        System.out.println("\n ** Message: This student is not exist.  ...");
        eventBusInterface.sendEvent(new Event(EventId.ClientOutput, "This student is not exist."));
    }
```

넘어온 정보를 학생 ID/과목 ID로 나눠준 다음에 학생 ID를 사용하여 해당 학생 ID가 학생 리스트에 존재하면 로그인에 성공했다는 메시지를 출력한다. 이후 학생이 수강 신청하고자 하는 과목 ID를 사용하여 학생이 이 과목을 수강했는지 하지 않았는지를 확인한다. 들었으면 이미 들었다는 메시지를 출력한 후 초기화면으로 돌아가고 듣지 않았으면 CourseMain.java로 이벤트와 학생 ID + 과목 ID로 이벤트가 넘어간다.

Components.Course :: CourseMain.java

```
} else if (event.getEventId() == EventId.Registeration) {
    String courseInfo = event.getMessage(); // 학생 + 과목 정보
    String courseCode = courseInfo.substring(courseInfo.length()-5,courseInfo.length()); // 과목ID
    if (coursesList.isRegisteredCourse(courseCode) == true) { // 있는 과목이면
        System.out.println("\n ** Message: Please, wait.  ...");
        eventBusInterface.sendEvent(
                new Event(EventId.SuccessRegistration , courseInfo)); // 학생 + 과목 정보
    } else {
        System.out.println("\n ** Sending an event(ID:" + EventId.ClientOutput + ") with");
        System.out.println("\n ** Message: This course is not exist.  ...");
        eventBusInterface.sendEvent(new Event(EventId.ClientOutput, "This course is not exist."));
```

CourseMain.java로 넘어오면 여기서는 과목 ID를 사용하여 이 과목이 현재 과목 리스트에 존재하는지를 확인한다. 만약 존재하지 않으면 이 과목은 존재하지 않는다는 메시지를 출력하고 존재하는 과목이면 잠시 기다리라는 메시지를 띄운 후 다시 StudentMain.java로 이벤트와 정보를 넘겨준다.

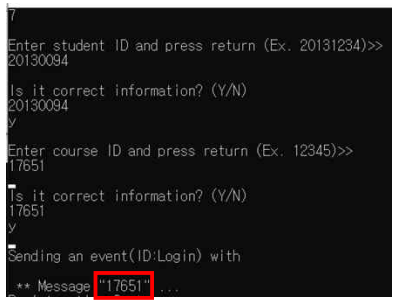
Components.Student :: StudnetMain.java

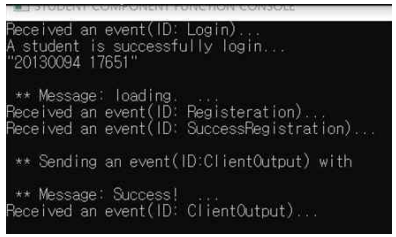
```
} else if (event.getEventId() == EventId.SuccessRegistration) { // 과목이 존재하고 학생이 듣지 않은 과목이면
    String studentInfo = event.getMessage(); // 추가하고자 하는 과목
    String student = studentsList.readStudent(studentInfo.substring(0, 8));
    boolean isSuccess = false;
    // 선수과목 들었는지 확인
    if (studentInfo.substring(9).equals("17651")) {
        if (student.contains("12345")) { // 선수과목 12345 들었으면
            student = student + " " + studentInfo.substring(9); // 추가
            studentsList.vStudent.remove(studentsList.getStudentIndex(studentInfo.substring(0, 8)));
            studentsList.vStudent.add(new Student(student));
            isSuccess = true;
        } else {
            System.out.println("Please take the required courses. ...");
            eventBusInterface.sendEvent(new Event(EventId.ClientOutput,
                    "Please take the required courses. ..."));
        }
    } else if (studentInfo.substring(9).equals("17652")) {
        if (student.contains("23456")) { // 선수과목 23456 들었으면
            student = student + " " + studentInfo.substring(9); // 추가
            studentsList.vStudent.remove(studentsList.getStudentIndex(studentInfo.substring(0, 8)));
            studentsList.vStudent.add(new Student(student));
            isSuccess = true;
        } else {
            System.out.println("Please take the required courses. ...");
            eventBusInterface.sendEvent(new Event(EventId.ClientOutput,
                    "Please take the required courses. ..."));
        }
    } else if (studentInfo.substring(9).equals("17654")) {
        if (student.contains("17651") && student.contains("12345")) { // 선수과목 17651 12345
            // 들었으면
            student = student + " " + studentInfo.substring(9); // 추가
            studentsList.vStudent.remove(studentsList.getStudentIndex(studentInfo.substring(0, 8)));
            studentsList.vStudent.add(new Student(student));
            isSuccess = true;
        } else {
            System.out.println("Please take the required courses. ...");
            eventBusInterface.sendEvent(new Event(EventId.ClientOutput,
                    "Please take the required courses. ..."));
        }
    } else if (studentInfo.substring(9).contains("17655")) {
        if (student.contains("17651") && student.contains("12345")) { // 선수과목 12345 17651 들었으면
            student = student + " " + studentInfo.substring(9); // 추가
            studentsList.vStudent.remove(studentsList.getStudentIndex(studentInfo.substring(0, 8)));
            studentsList.vStudent.add(new Student(student));
            isSuccess = true;
        } else {
            System.out.println("Please take the required courses. ...");
            eventBusInterface.sendEvent(new Event(EventId.ClientOutput,
                    "Please take the required courses. ..."));
        }
    } else if (studentInfo.substring(9).equals("12345") || studentInfo.substring(9).equals("23456")
            || studentInfo.substring(9).equals("17653")) { // 선수과목 없는 과목을 들었으면
        student = student + " " + studentInfo.substring(9); // 추가
        studentsList.vStudent.remove(studentsList.getStudentIndex(studentInfo.substring(0, 8)));
        studentsList.vStudent.add(new Student(student));
        isSuccess = true;
    }
    if (isSuccess == true) {
        System.out.println("\n ** Sending an event(ID:" + EventId.ClientOutput + ") with");
        System.out.println("\n ** Message: Success!  ..." + student);
        eventBusInterface.sendEvent(
                new Event(EventId.ClientOutput, "Success!!\n\r" + student));
    }
```

다시 이벤트가 StudentMain.java로 넘어오면 이제 해당 학생이 선수과목을 들었는지를 확인해준다. 선수과목을 잘 들었으면 수강 신청에 성공하여

해당 학생에 과목을 추가해주고 성공했다는 메시지와 그 결과를 출력한다. 하지만 만약 선수과목을 다 듣지 않았으면 선수과목을 들어달라는 메시지와 함께 수강 신청은 실패하며 초기화면으로 돌아간다.

(3) 결과





======== New event has been arrived ========
Success!!
20130094 Jang Goyoung CS 12345 23456 17653 17651

- Output 결과 (수강 신청한 학생 결과)

<수강 신청 완료 후 학생 리스트>

```
** Message:
20100123 Hwang Myunghan CS 12345 23456 17651 17652
20090421 Kim Jason EE 17651 17652 17653 17654
20110512 Park Hongsun CS 12345 23456 17652 17654
20100123 Kim Yunmi CS 12345 17651 17652 17653
20100125 Kim Hokyung ME 12345 23456 17651 17654
20100323 Jung Philsoo CS 12345 23456 17651 17655
20080678 Ahn Jonghyuk EE 12345 17651 17652 17654
20110298 Lee Mijung ME 12345 23456 17651 17652
20120808 Kim Sungsuk EE 12345 23456 17651 17652
20080603 Park Kitea ME 17651 17652 17653
20070452 Ko Kyungmin CS 12345 17652 17655
20130091 Kim Chulmin CS 12345 23456 17651 17652
20110876 Park Kiyoung EE 12345 17651 17652 17654
20100128 Kim Minsu CS 12345 17651 17652 17653
20100131 Kim JungMi CS 17651 17652 17653 17655
20130094 Jang Goyoung CS 12345 23456 17653 17651
```

<수강 신청 시, 잘못된 학생 ID를 입력했을 경우>

```
7
Enter student ID and press return (Ex. 20131234)>>
123

Is it correct information? (Y/N)
123
y

Enter course ID and press return (Ex. 12345)>>
12345

Is it correct information? (Y/N)
12345
y

Sending an event(ID:Login) with

 ** Message "12345" ...
```

COURSE REGISTRATION SYSTEM OUTPUT CONSOLE

```
Received an event(ID: ClientOutput)...
```

COURSE COMPONENT FUNCTION CONSOLE

```
Received an event(ID: Login)...
Received an event(ID: ClientOutput)...
```

STUDENT COMPONENT FUNCTION CONSOLE

```
 ** Sending an event(ID:ClientOutput) with

 ** Message: This student is not exist.  ...
Received an event(ID: ClientOutput)...
```

```
======== New event has been arrived ========
This student is not exist.
```

- 학생이 존재하지 않는다는 메시지 출력

<수강 신청 시, 잘못된 과목 ID를 입력했을 경우>

```
7
Enter student ID and press return (Ex. 20131234)>>
20130094

Is it correct information? (Y/N)
20130094
y

Enter course ID and press return (Ex. 12345)>>
123

Is it correct information? (Y/N)
123
y

Sending an event(ID:Login) with

 ** Message "123" ...
```

STUDENT COMPONENT FUNCTION CONSOLE

```
Received an event(ID: Login)...
A student is successfully login...
"20100123 123"

 ** Message: loading.  ...
Received an event(ID: Registeration)...
Received an event(ID: ClientOutput)...
```

COURSE REGISTRATION SYSTEM OUTPUT CONSOLE

```
Received an event(ID: Login)...
Received an event(ID: Registeration)...
Received an event(ID: ClientOutput)...
```

COURSE COMPONENT FUNCTION CONSOLE

```
Received an event(ID: Login)...
Received an event(ID: Registeration)...

 ** Sending an event(ID:ClientOutput) with

 ** Message: This course is not exist.  ...
Received an event(ID: ClientOutput)...
```

```
======== New event has been arrived ========
This course is not exist.
```

- 과목이 존재하지 않는다는 메시지 출력

(4) 느낀점

문제 B를 해결할 때에는 여러 가지 예상치 못한 오류가 많아서 당황했었다. 일단 프로그램이 좀 더 무거워지다보니 노트북이 느려지는 것에 그치지 않고 아예 멈추는 일도 발생하였고, 노트북이 버티지 못할까봐 넣었던 thread.sleep()의 영향으로 프로그램이 돌아가지 않는 오류도 발생하였다. 처음에는 thread.sleep()의 영향인 줄 모르고 다른 곳에서 원인을 찾다가 시간을 많이 허비하였다. 하지만 의외의 장점도 발견할 수 있었다. bat파일을 실행한 후 코드에 에러가 있으면 해당 cmd가 자동으로 꺼지게 되는데 컴퓨터의 렉으로 인하여 오류에 대한 내용을 확인할 수 있어서 문제를 해결하기에 더 용이하였다.

3. 결론

1) 2개의 아키텍처 장단점 비교

	장점	단점
Pipes & Filters	1. 필터 교환/조합에 유연하다. 2. 다른 필터와의 종속성이 없으므로 필터 컴포넌트를 재사용할 수 있고, 수정을 최소화할 수 있다.	1. 상태 정보를 공유하게 되면 비용이 많이 들며 유연성을 저하시키는 요인이 된다. 2. 데이터 변환에 과부하가 발생한다.
Event-Bus	1. 상/하위 관계를 가지고 그 안에서만 데이터를 주고받을 수 있다는 그런 기본적인 규칙을 지키지 않아도 된다. 2. 규모가 크고 고도로 분산화된 시스템에 효과적이다.	1. Component가 많아지면 이들을 모두 관리하는 데에 있어서 난이도가 상승한다. 2. 메모리 수요가 증가할 수 있다.