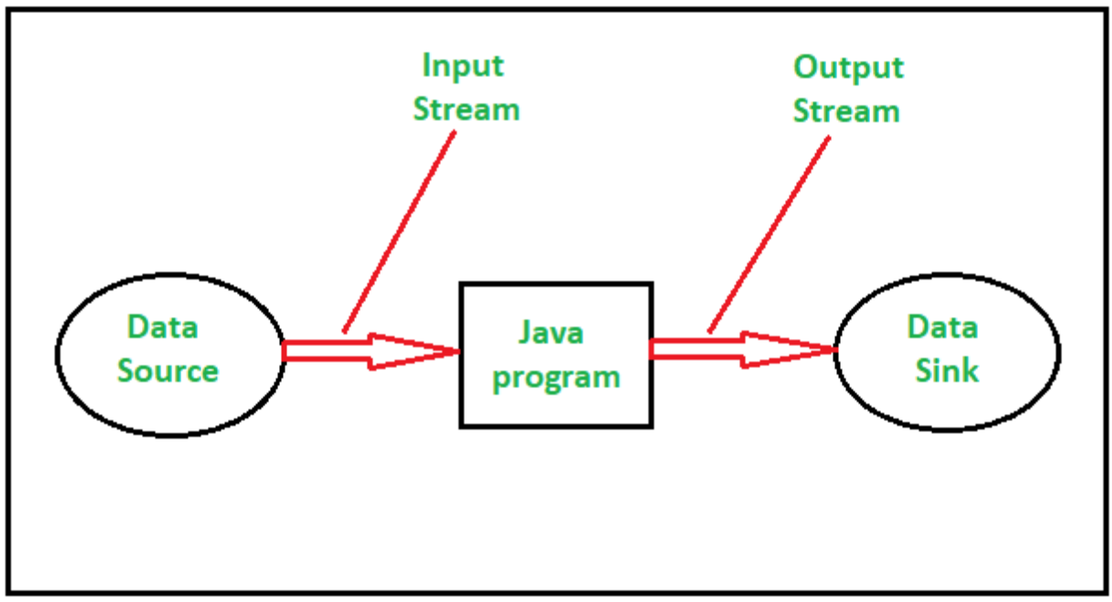
2025-09-04 Summary

IO

- 프로그램이 외부세계와 상호작용(input, output) 하는 핵심기능



두 가지 분류

1. 장식(Decorator)

```
다양한 방식으로 읽고 쓰는 메소드를 가짐
기본 기능을 확장하거나 개선
InputStream, OutputStream, Reader, Writer를 생성자에서 받아들임
```

2. 주인공(주 구성요소)

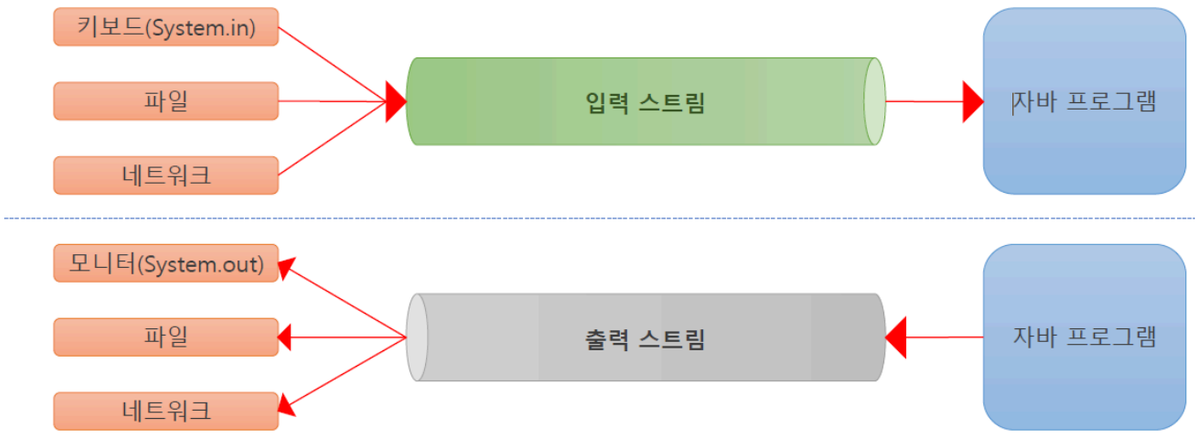
```
데이터의 근원지, 목적지에 따른 클래스
```

-> 구분 방법 : 장식은 다른 IO객체를 생성자 매개변수로 받는다

---

1. 스트림(Stream)

- 데이터가 흐르는 통로를 추상화한 개념
  = 데이터는 스트림을 통해 이동



- **단방향**으로만 흐르므로, 입력하는 기능과 출력하는 기능 둘 다 필요!
- 한 문자씩 읽고, 한번 읽었으면 끝!

분류

- Java I/O는 4개의 *추상 클래스* 기반

종류	about
InputStream	바이트 단위 입력
OutputStream	바이트 단위 출력
Reader	문자(2byte char)단위 입력
Writer	문자(2byte char)단위 출력

---

2. 바이트 스트림 입출력

- 모든 종류의 데이터를 처리할 수 있는 가장 기본적인 스트림
  - 이진 데이터 처리에 적합 (이미지, 오디오, 비디오)
  - 1바이트씩 처리
  - 버퍼 사용 시 효율 높아짐

주요 클래스

InputStream 계열

- FileInputStream : 파일에서 바이트 단위 읽기
- ByteArrayInputStream : 바이트 배열에서 읽기
- BufferedInputStream : 버퍼를 사용한 효율적인 읽기
- DataInputStream : 기본형 데이터 읽기

OutputStream 계열

- FileOutputStream : 파일에 바이트 단위 쓰기
- ByteArrayOutputStream : 바이트 배열에 쓰기
- BufferedOutputStream : 버퍼를 사용한 효율적인 쓰기
- DataOutputStream : 기본형 데이터 쓰기

```java
try (FileInputStream in = new FileInputStream("input.jpg");
     FileOutputStream out = new FileOutputStream("output.jpg")) {

    int byteData;
    // 파일 끝(-1 = 리턴값값)까지 한 바이트씩 읽고 쓰기
    while ((byteData = in.read()) != -1) {
        out.write(byteData);
    }
    System.out.println("파일 복사 완료!");

} catch (IOException e) {
    System.err.println("파일 처리 중 오류: " + e.getMessage());
}
```

-> `read()` 로 바이트 단위 읽기, `write()` 로 출력
-> try-with-resource 로 close 자동으로 처리

---

3. 문자 단위 입출력(Reader/Writer)

- 텍스트 처리에 최적화
- 인코딩 자동 처리
- 유니코드 지원 (다국어 가능함)

주요 클래스

Reader 계열

- FileReader : 파일에서 문자 단위 읽기
- StringReader : 문자열에서 읽기
- BufferedReader : 버퍼를 사용, readLine() 제공
- InputStreamReader : 바이트 -> 문자 변환

Writer 계열

- FileWriter : 파일에 문자 단위 쓰기
- StringWriter : 문자열에 쓰기
- BufferedWriter : 버퍼를 사용한 효율적인 쓰기
- OutputStreamWriter : 문자 -> 바이트 변환

```java
try (FileReader reader = new FileReader("input.txt");
     FileWriter writer = new FileWriter("output.txt")) {

    int character;
    // 파일 끝(-1 = 리턴값)까지 한 문자씩 읽고 쓰기
    while ((character = reader.read()) != -1) {
        writer.write(character);
    }
    System.out.println("텍스트 파일 복사 완료!");

} catch (IOException e) {
    System.err.println("파일 처리 중 오류: " + e.getMessage());
}
```

-> `read()` 는 문자를 반환, *EOF(End Of File : 파일의 끝을 표현하기 위해 정의해 놓은 상수)* 는 -1
-> 텍스트 파일 복사할 때 적합

---

3. 표준 입출력 (System.in, System.out, System.err)

- System.in : 표준 입력 (InputStream, Scanner/BufferedReader와 사용)
- System.out : 표준 출력 (PrintStream, 일반 메시지 출력)
- System.err : 표준 오류 출력 (PrintStream, 에러 메시지 분리 출력)

```
try (BufferedReader reader = new BufferedReader(
          new InputStreamReader(System.in))) {

    System.out.print("이름을 입력하세요: ");
    String name = reader.readLine();

    System.out.print("나이를 입력하세요: ");
    int age = Integer.parseInt(reader.readLine());

    System.out.println("안녕하세요, " + name + "님!");
    System.out.println("당신은 " + age + "살입니다.");

    if (age < 0) {
        System.err.println("오류: 나이는 음수일 수 없습니다.");
    }

} catch (IOException e) {
    System.err.println("입력 오류: " + e.getMessage());
}
```

-> 정상 출력은 `System.out` / 오류 출력은 `System.err`

---

4. Scanner

- 다양한 타입 입력을 쉽게 처리함
- nextInt(), nextDouble(), nextBoolean(), nextLine()
- 파일 입력도 가능 - `new Scanner(new File("input.txt"))`

```
Scanner sc = new Scanner(System.in);

System.out.print("정수를 입력하세요: ");
int num = sc.nextInt();

System.out.print("문자열을 입력하세요: ");
String str = sc.next();

System.out.println("입력된 값: " + num + ", " + str);

sc.close();
```

-> *버퍼에 개행(\n)이 남아있으므로 nextLine() 주의!*

하지만 next()는 공백/개행을 건너뛰고 다음 토큰(문자열)을 읽으므로 괜찮음

5. 파일 입출력과 File 클래스

File 클래스 기능

- 파일/디렉토리 경로, 크기, 권한, 이름, 삭제 여부 등을 관리
- 실제 파일 생성X, 파일 객체(경로 정보)만 관리
- 디렉토리 도 파일로서 취급!
- exists() : 존재 여부 확인
- canRead(), canWrite() : 권한 확인
- getAbsolute(), getCanonicalPath() : 경로 확인 *(절대/상대)*
- getName(), getParent() : 이름/부모 디렉토리 확인

```
File f = new File("example.txt");

if (f.exists()) { // 파일이 존재할 때
    System.out.println("length: " + f.length()); //크기
    System.out.println("canRead: " + f.canRead());
    System.out.println("canWrite: " + f.canWrite());
    System.out.println("AbsolutePath: " + f.getAbsolutePath());
    System.out.println("CanonicalPath: " + f.getCanonicalPath());
    System.out.println("Name: " + f.getName());
    System.out.println("Parent: " + f.getParent());
    System.out.println("Path: " + f.getPath());
} else {
    System.out.println("파일이 존재하지 않습니다.");
}
```

-> 파일 존재 여부, 크기, 읽기/쓰기 권한, 경로 정보 등 조회 가능

& 파일 삭제 예제

```
File f = new File("example.txt");

if (f.exists()) {
    boolean deleted = f.delete();   // 삭제 시도
    if (deleted) { //boolean값 return
        System.out.println("파일 삭제 성공");
    } else {
        System.out.println("파일 삭제 실패");
    }
} else {
    System.out.println("파일이 존재하지 않습니다.");
}
```

-> `delete()` 는 성공 여부를 boolean으로 반환

& 디렉토리 목록 조회 예제

```
File dir = new File("myFolder");

if (dir.isDirectory()) {
    File[] files = dir.listFiles();
    for (File file : files) {
        System.out.print(file.getName() + "\t");
```

```
            if (file.isDirectory()) {
                System.out.println("[디렉토리]");
            } else {
                System.out.println("[파일] 크기: " + file.length());
            }
        }
    } else {
        System.out.println("디렉토리가 아닙니다.");
    }
```

-> 디렉토리 여부 확인 후, `listFiles()` 로 하위 파일, 폴더 탐색

& 임시 파일 생성과 삭제 예제

```
File temp = File.createTempFile("tmp_", ".dat");
System.out.println("임시 파일 생성: " + temp.getAbsolutePath());

// JVM 종료 시 자동 삭제
temp.deleteOnExit();
```

-> `createTempFile()` 로 임시파일 생성, `deleteOnExit()` 로 JVM 종료 시 자동 삭제

파일 입출력 과정 (일반 절차)

읽기 과정

1. 파일 연결 (FileInputStream / FileReader)
2. read()로 데이터 읽기
3. close()로 리소스(자원) 해제

쓰기 과정

1. 파일 연결 (FileOutputStream / FileWriter)
2. write()로 데이터 읽기
3. close()로 리소스(자원) 해제

파일 복사(바이트 단위) 예제

```
try (FileInputStream fis = new FileInputStream("source.pdf");
     FileOutputStream fos = new FileOutputStream("dest.pdf");
     BufferedInputStream bis = new BufferedInputStream(fis);
     BufferedOutputStream bos = new BufferedOutputStream(fos)) {

    byte[] buffer = new byte[1024];
    int bytesRead;

    // 버퍼 단위로 읽고 쓰기
    while ((bytesRead = bis.read(buffer)) != -1) {
        bos.write(buffer, 0, bytesRead);
    }

    System.out.println("파일 복사 완료!");

} catch (IOException e) {
```

```
        System.err.println("파일 복사 실패: " + e.getMessage());
    }
```

-> buffer 단위 읽기/쓰기 = 성능 UP!

최형규님

오늘 정말 다사다난했다. 도서관에서 분실 문제가 생겨서 경찰도 만나고.. 다행히 다찾았고 우산만 잃어버렸지만, 그래서 대피한 카페는 와이파이 문제가 생기다가, 네트워크 드라이버가 날아가고 다시 초기화 시키고,

그래도 수업 자리비움 표시는 해도 폰으로 라디오처럼 듣다보니 그래도 오늘 배운것들 개념은 좀 이해할 수 있어서 감사했다. 후에 오늘 배운것 자신이 작성한것들 교수님들한테 자세하게 코드리뷰 부탁드렸었는데, 흔쾌히 해주셨고

오늘 배운것들 전반적으로 놓쳤던 부분 필요한 것들 채울 수 있어서 감사했다. 후에 교안이랑 필기한것들 보며 또 내것으로 만들어야겠다.

자바 일단 1차적인 진도 다나갔는데, 다 뿌수자!

오늘의 키워드

- 솔리드원칙
- IO
- 데코레이터 패턴
- 바이트단위 입출력
- 인풋스트림 vs 아웃풋스트림
- 버퍼
- 시스템
- 직렬화
- ois, oos

백종현님

오늘은 자바I/O에 대해 깊게 살펴보는 시간을 가졌습니다.
오늘은 실습을 많이 했는데 실습을 하며 코드를 짜는 시간이 재미있었습니다.
입출력 방식은 정말 많은 방식이 있는것 같습니다.
정말 많은 방식중 자신에게 필요한 방법을 잘 찾아보고 잘 조립하여 활용하는게 가장 중요한 요점인것 같습니다.
드디어 자바 진도의 끝이 보여갑니다.
화이팅!
오늘의 키워드

- I/o
- 스트림
- 조립

윤수정님

오늘은 내 부족한 이해력을 실감한 하루였다. 분명 강의를 들으며 따라가고는 있지만, 개념들이 머릿속에 오래 남지 않고 스쳐 지나가 버리는 것이 아쉬웠다. 한 번에 모든 내용을 이해하는 것이 쉽지 않다는 건 알지만, 그래도 계속 흘려보내고 있다는 느낌이 마음을 무겁게 했다.

그러나 오늘은 내가 요약 담당이기에, 단순히 아쉽다고만 남기지 않고 적극적으로 정리해보려 한다. 교안과 강사님의 코드를 꼼꼼히 살펴보고, 단순히 '아는 것'에서 끝나는 것이 아니라 '남에게 설명할 수 있을 정도'로 내 두뇌에 각인시키는 것이 목표다. 이해가 부족했던 순간들을 채워 넣으며, 내일은 조금 더 단단한 기반 위에서 다시 출발할 수 있기를 바란다.

김도은님

예외 처리는 단순히 에러 잡기가 아니라, 계층 간 의미 있는 신호 전달 수단이다.
I/O 스트림과 버퍼는 속도와 편리함에서 큰 차이를 만든다. → 대용량 처리 시 Buffered 계열이 필수.
직렬화는 객체를 파일이나 네트워크로 쉽게 보낼 수 있는 방법이지만, 보안(transient)과 버전 관리(serialVersionUID)를 반드시 고려해야 한다.
스트림은 파이프, 버퍼는 물통이라는 비유 덕분에 개념이 명확해졌다.

남혜린님

I/O 스트림을 배우면서부터 코드 리뷰가 점점 어려워지기 시작했다. 이해는 되지만 적용하는 과정에서 막히는 부분이 많아 조금은 답답하다. 그래도 금요일까지는 최대한 열심히 수업을 듣고, 주말과 월요일까지 보강하면서 다음 주를 힘차게 달릴 준비를 해야겠다.

점심시간이나 회고 시간쯤만 되면 반복적으로 발생하는 서버 문제 때문에 집중이 자꾸 흐트러져서 아쉽지만, 빠르게 개선되길 바란다. 작은 스트레스에 크게 휘둘리지 않고, 오늘도 남은 공부를 이어가야겠다.

진솔빈님

I/O, stream, buffer에 대해서 개인적으로 덧붙이며 공부해봤다.

파일 입출력(I/O)를 '수도관'에 비유할 수 있겠다.

거기에 stream 으로 파일과 프로그램 사이 데이터 통로 , 데이터를 한 방향으로만 보내는 '파이프' 역할을 하는 "FileInputStream"과 "FileOutputStream"이 있고, 각 데이터의 흐름 방향은 FileInputStream이 파일에서 프로그램으로, 파일을 읽기 위한 통로이며 FileOutputStream이 프로그램에서 파일로, 파일을 쓰기 위한 통로로 역할한다.

Buffer는 스트림을 통해 흐르는 데이터를 임시로 담는 '바구니' 역할을 한다.

한 바이트씩이 아닌, 일정량을 옮긴다. 데이터를 한 곳에서 다른 곳으로 전송하는 동안 일시적으로 그 데이터를 보관하는 곳이다. '버퍼링'이라는 표현 역시 버퍼에서 나온 것인데, 버퍼를 채우는 동작을 말하는 것이다. 버퍼를 Queue라고도 말한다.

버퍼I/O가 없으면 읽기 또는 쓰기 요청은 OS에서 직접 처리된다. 이게 덜 효율적이고, 결과적으로 '오버헤드'를 일으키므로 이걸 줄이기 위해 자바 플랫폼에서는 버퍼링된 I/O 스트림을 구현한다.

즉, "바구니에 담아 옮기기"가 OS단에서 직접 처리되는것보다 데이터를 효율적으로 처리하는 표준 방법으로 쓰인다.

byte[] bytes = new byte(1024) 는 byte라는 바구니를 1024바이트의 크기로 만드는 것이고, 이것이 버퍼역할을 하게 된다.(임시 저장소)

while(n= fis.read(bytes != -1) -> 파일의 끝(End Of File)에 read 메서드가 도달하면, 더이상 데이터가 없음을 감지하며 -1을 반환하기 전까지만 while문을 돌려 반복 탐색하라는 뜻. 파일의 끝에 도달하지 않는 한 계속 데이터를 읽고 처리

일단, 그 외의 부분도 있지만, 조금 헷갈렸던 개념에 대해서 간단히 쉬운말로 풀이해봤다.

지금 이 과정에서 공부하며 대부분의 시간을 이렇게 나만의 언어로 쪼개서 이해시키고 소화하는 과정에 대부분을 할애했다. 그나마, 프로젝트를 만들면서 더 직관적으로 와닿긴 했지만, 아직까진 이런 작업을 해주며 계속 내가 먹을 수 있는 먹이상태로 만들고 먹어야 일단 다음챕터로 넘어갈 수 있다.

확실히, 많은 후기에서 봤듯, 진입장벽이 있는 것은 사실이지만,
다행히 담주 월요일도 쉬고, 그 때를 맞이해서 프로젝트 작업(why 이 코드를 그 자리에 썼나?)을 기반으로 개념도 좀 다져가면 나아질듯 하다.

규칙적인 생활과 운동이 이 훈련기간을 버티게 해주니, 꾸준한 기본체력이 중요함을 더 느낀다. 할 게 많다.