



어제 하던 예외처리
에러들 중 예외에 처리되는 부분 `uncheckedException`
`try`: 예외가 발생 할 것 같은 부분에 작성
`catch`: 만약 발생시 `Exception` 처리
`final`: 예외가 나든 말든 한번은 실행.
문서를 보고 예외처리 할지 말지 판단
`catch` 경우를 많이 적을 수 있지만
상위 예외를 우선에 둘경우 하위 예외가 처리 안된다.

Throws : 호출한 메소드에게 예외를 던짐


호출한 메소드에게 예외를 처리하라고 위임하는 행위
호출한 곳으로 예외를 던진다
`try catch`를 여러개 쓰는것 보다 한곳에서 한번에 처리하면 좋으니깐

 `e.printStackTrace();` 처리된 예외 추적.
`RuntimeException` : 개발중 흔하게 일어나는 실수도.
• `if`문으로 대부분을 처리할 수 있다.
`UncheckedException`
• `RuntimeException`과 자식 예외들을 지칭함
• 프로그램 수행 중 개발자의 코딩 실수 혹은
사용자의 잘못된 값 입력으로 흔하게 발생할 수 있는 예외
◦ 예외처리 구문이 아니라 `if`문 같은 단순 코드로 해결 가능.
◦ `if`가 더 속도 나는 경우 높음
`checkedException`
• `RuntimeException`과 자식 예외를 제외한 나머지 예외
• `if`문같은 단순 코드로 해결 불가능하여 반드시 예외처리 구문 필요

`Exception`의 범위는

상속이 예외를 조금더 구체적이거나 동일해야 한다
위에 사진은 입출력의 모든 에러에서 파일의 입출력 에러로 구체화 한 것
`child` 클래스에 예외로 최상위 부모 예외인 `Exception`을 대입할 시 에러가 뜬다.

Collection : 자바에서 제공하는 자료구조 모음

`List` , `Set` , `Map`

 11_컬렉션.pdf 658 kB

객체를 하나의 문자열로 = toString

자료구조 : 데이터(자료)를 메모리에서 구조적으로 처리하는 방법론.
배열의 단점
컬렉션의 장점
• 저장하는 크기의 제약이 없다.
• 추가 , 삭제 , 정렬 등의 기능 처리가 간단하게 해결된다.
◦ 자료를 구조적으로 처리하는 자료구조가 내장되어 있어 알고리즘 구현이 필요 없다
• 여러 타입의 데이터가 저장 가능하다.
◦ 객체만 저장할 수 있기 때문에 필요에 따라 기본 자료형을 저장해야 하는 경우 `Wrapper` 클래스 사용

인터페이스
• 관련 없는 것들을 연결
• 공통된 메소드명을 제공해서 규약을 만든다.
• 추상메소드 : 상속받은 자식이 오버라이딩 해야 한다.
• 모든 자식 클래스가 같은 이름의 메서드를 가짐



List / Set / Map 은 인터페이스.

List :

- 배열 기반 목록 (배열의 모양과 비슷하다)
- 중복이 허용된다.
- 순서가 있다

Set :

- 집합 (순서를 파악하지 못하기 때문에 중복을 제거한다)
- 중복이 허용되지 않는다
- 순서가 없다

Map :

- 지도와 비슷하다
- 출석부 번호를 보고 누구인지 알아낸다
- 키의 값과 쌍으로 저장 (키 : 주소 / 키를 보고 값을 확인함 , 주소를 보고 집을 보러 감)
- 키는 중복 저장이 안됨 (주소가 중복되면 안된다.)

Collection 인터페이스 : **List , set** 공통 코드를 작성한 단순한 부모 인터페이스

JAVA Collection : 자바의 자료구조 모음(**List, Set ,Map**);

ArrayList(용량)



< 원하는 타입 > 을 적어 원하는 객체 타입만 대입하게 한다.

리스트를 생성 후 활용을 해보았다

하위 예제는 11_collection Project에 StudentService를 기반으로 해석했다.

```
private List<Student> studentList = new ArrayList<Student>();
```

접근 제한자가 Private인 List 형식의 studentList 를 컬렉션 중 배열과 비슷한 ArrayList를 참조해서 선언한다
이때 <제너럴>로 대입 가능한 자료형은 Student만 가능하게 허락했다.

```
studentList.add(new Student());
```

위에서 선언된 StudentList에 새로운 객체를 추가한다

이 때 새로운 객체는 (Object를 상속받은 객체면 가능하다, 사실상 모든 객체를 매개변수로 전달 가능 하다는 걸 의미)

위의 상황에서는 studentList에 Student(학생의 정보들이 모여있는 value object)를 대입한다

이는 학생의 정보들이 studentList에 순서대로 대입 된다는 걸 의미한다.

```
System.out.println(studentList.get(0).getName());
```

studentList에서

get (0): 0번째 배열에서 가져온다

getName() 해당 배열에서 Name이라는 필드 값을 가져온다

```
studentList.add(new Student(name, age, region, gender, score))
```

studentList에 추가한다 (괄호) 안에 매개 변수를

현재 studentList에는 <Student> 자료형만 추가가 가능하기 때문에

각각의 매개변수들은 Student의 매개변수 생성자로 set한다

```
studentList.isEmpty()
```



`studentList`라는 `List`의 길이를 확인해 한 명도 없을 경우 "학생 정보가 없다"라는 내용을 출력하고 싶다

1) `List.size`로 `List`의 길이를 0보다 작거나 같은 경우 출력 하는 방법이 있다

2) `studentList.isEmpty()` 문법을 사용해 훨씬 간결하게 `List`의 값이 있는지 없는지 확인 할 수 있다.

`isEmpty()`: 이름 그대로 앞쪽에 위치한 `List`가 비어있을 경우 `True`;

`for (Student std : studentList) {`

개인적으로 상당히 난해했다. `for`문법의 향상된 버전

`for`(오른쪽의 값을 저장할 변수 : 컬렉션 또는 배열명)

ex



위의 경우에는 `Student`를 참조한 `std` 변수를 생성해서 `studentList`의 값을 대입한다.

`for`문을 반복하면서 `studentList`의 배열을 0, 1, 2, 3 순차 접근한다.

`print` 문으로 몇 번 `List`인지 체크 시켜준다.

그 다음 0부터 순차 접근 중인 `studentList`의 데이터를 `std`에 대입해서 출력한다.

이때 `std`의 값이 주소 값이 아닌 데이터로 출력 되는 것은 내가 `Student`클래스에서 오버라이딩을 해놨기 때문이다.



`toString`이 오버라이딩 되어있으면 주소 값이 아니라 오버라이딩된 값이 나온다
위의 `eachFor`문에서 `std`이후 별도의 순서를 표기 안하는 이유는 해당 `for`문법에서는
자동적으로 목록의 초기값 부터 끝까지 순차적으로 올라가기 때문이다.

`studentList.size()`

보기처럼 간단하게 해당 컬렉션의 사이즈를 보여준다.
해당 컬렉션의 길이가 0, 1, 2, 3 이면 4가 출력 된다

`if (studentList.isEmpty())`

만약 (컬렉션이 .비어 있다면())이라는 의미
비어있을 경우 `true`을 값을 가진다.

`System.out.println(studentList.get(index));`

`studentList`의 입력 받은 `index`번의 값을 가져와(`get`) `toString`한다.

`Student temp = studentList.set(index, new Student(name, age, region, gender, score));`

입력 받은 `index`번째에 새로운 학생 정보를 세팅 == 수정

이때 `index`번째 있던 기존 학생 정보가 반환 된다.

이 말이 좀 헷갈렸는데 다시 곱씹어 보면 어렵지 않다

1. `new Student(name, age, region, gender, score)`의 값을
2. `index` 번 `studentList`에 `set`으로 대입한다
3. `studentList.set(index, new Student(name, age, region, gender, score));`이 구문이 실행되면
4. `studentList(index)`에는 새로 입력된 값이 수정되어 대입 되고 원래 있던 값은 밖으로 반환 되는데
5. 그때 이 반환 되는 값을 `Student` 참조자료형 `temp`에 저장 시킨 것이다.

그런 다음

6. `return temp.getName() + "의 정보가 변경 되었습니다.";`를 반환하는데
7. 이때 `temp`에는 수정되기 전의 값이 반환 되어 저장되어 있으니 출력하게 되면 수정되기 전의 이름이 출력된다

특정 컬렉션의 데이터 삭제



- 위쪽 `if`문들에서 예외처리 범위를 벗어난 검색 들을 삭제한다.
- 여기서 `Y` 혹은 `N`을 입력 받는데 소문자를 적는 경우가 있는데
이 경우 `toUpperCase()`으로 소문자를 대문자로 변환해준다

- 만약 모든 예외 상황들이 처리되고 정상 처리되면 입력 받은 `index`번의 `studentList`를 `remove`한다
- `remove`: 삭제한다. 이때에도 바로 위에 상황 처럼 `Student`형 참조변수 `temp`에 삭제되기 전의 값이 저장된다

`input.equals(std.getName())`

입력 받은 값 `input`은 문자열 이기 때문에 `.equals`로 비교한다
이 구문은 `for`문으로 반복하면서 컬렉션의 모든 값을 비교해 본다.
※입력 받은 값이 정확히 일치해야 한다.

`std.getName().contains(input)`

`equals` 비교문과 다른점은 완벽히 일치 하지 않아도 `input`값을 포함하기만 하면 된다
`.contains`: 포함한다
ex) `std.getName("김현기").contains(현)` 일 경우 "현"이라는 글자가 포함되기 때문에 값이 출력된다