0307

스캐너의 동작 원리 + 문제점 + 해결방법
스캐너의 동작 원리 + 문제점 + 해결방법

1) 동작 원리
- next(); nextXXX(); 에서 next(다음)의 의미
스캐너를 이용한 입력 시, 입력 값이 바로 프로그램으로 전달 되는 것이 아닌
입력 버퍼라는 곳에 임시 저장이 된다.
(Buffer : 데이터를 모아서 한 번에 이동하는 용도의 저장 공간(==바구니))
(입력 버퍼 : 키보드로 입력된 모든 문자를 묶어서 저장하는 임시 공간
+ 띄어쓰기, 엔터도 저장함.)

ex) [키보드 입력] : 1234(엔터) -> [입력 버퍼] : 1234(엔터)

2) 문제점
* next() (또는 nextInt(), nextDouble())
- 입력 버퍼에서 다음 공백 문자(띄어쓰기, 엔터)를 만나기 전 까지의 문자열을 읽어옴
단, 입력 버퍼 맨 앞 공백 문자는 무시

ex) [키보드 입력] : abc(엔터) , [입력 버퍼] : abc(엔터1)
sc.next(); "abc" , [입력 버퍼] : (엔터1)

[키보드 입력] : def(엔터) , [입력 버퍼] : (엔터1)def(엔터2)
sc.next(); "def" (엔터1 무시), [입력 버퍼] : (엔터2)

**** 무조건 입력 버퍼에 (엔터)가 하나씩 남음!! ****

* nextLine()
- 입력 버퍼에서 다음 한 줄 == 다음 (엔터) 까지의 문자열을 읽어옴.
--> 제일 앞에 있는 공백문자를 무시할 수 없음!

ex)

[키보드 입력] : abc(엔터) , [입력 버퍼] : abc(엔터)
sc.nextLine(); "abc" , [입력 버퍼] : (텅 빔)

!문제 상황!
[키보드 입력] : abc(엔터) , [입력 버퍼] : abc(엔터)
sc.next(); "abc" , [입력 버퍼] : (엔터)

[키보드 입력] : def(엔터) , [입력 버퍼] : (엔터)def(엔터)
sc.nextLine(); "" (빈칸) , [입력 버퍼] : def(엔터)

**** 입력 버퍼 맨 앞에 개행문자(엔터)가 남아 있어서
nextLine() 시 빈칸이 읽어와지는 문제는 발생!!! ****
}

1) 동작 원리
- next(); nextXXX(); 에서 next(다음)의 의미
스캐너를 이용한 입력 시, 입력 값이 바로 프로그램으로 전달 되는 것이 아닌
입력 버퍼라는 곳에 임시 저장이 된다.
(Buffer : 데이터를 모아서 한 번에 이동하는 용도의 저장 공간(==바구니))
(입력 버퍼 : 키보드로 입력된 모든 문자를 묶어서 저장하는 임시 공간
+ 띄어쓰기, 엔터도 저장함.)

ex) [키보드 입력] : 1234(엔터) -> [입력 버퍼] : 1234(엔터)

2) 문제점
* next() (또는 nextInt(), nextDouble())
- 입력 버퍼에서 다음 공백 문자(띄어쓰기, 엔터)를 만나기 전 까지의 문자열을 읽어옴
단, 입력 버퍼 맨 앞 공백 문자는 무시

ex) [키보드 입력] : abc(엔터) , [입력 버퍼] : abc(엔터1)
sc.next(); "abc" , [입력 버퍼] : (엔터1)

[키보드 입력] : def(엔터) , [입력 버퍼] : (엔터1)def(엔터2)
sc.next(); "def" (엔터1 무시), [입력 버퍼] : (엔터2)

**** 무조건 입력 버퍼에 (엔터)가 하나씩 남음!! ****

```
    * nextLine()
    - 입력 버퍼에서 다음 한 줄 == 다음 (엔터) 까지의 문자열을 읽어옴.
      --> 제일 앞에 있는 공백문자를 무시할 수 없음!

    ex)

    [키보드 입력] : abc(엔터)          , [입력 버퍼] : abc(엔터)
    sc.nextLine();  "abc"           , [입력 버퍼] :  (텅 빔)



    !문제 상황!
    [키보드 입력] : abc(엔터)          , [입력 버퍼] : abc(엔터)
    sc.next();     "abc"            , [입력 버퍼] : (엔터)

    [키보드 입력] : def(엔터)          , [입력 버퍼] : (엔터)def(엔터)
    sc.nextLine();  "" (빈칸)        , [입력 버퍼] : def(엔터)

    **** 입력 버퍼 맨 앞에 개행문자(엔터)가 남아 있어서
         nextLine() 시 빈칸이 읽어와지는 문제는 발생!!! ****
}
```

객체지향 언어

클래스를 new해서 객체화 하는 것을 = 인스턴스화 한다

Class

규칙
[접근 제한자]
~ (틸드 , tilde)
클래스에서 변수와 관련된 속성을 적는 곳을 필드라 한다.
같은 타입의 여러 객체가 공유할 목적의 필드에 사용한다.
Student std1= new Student();

Student라는 자료형을 참조할 변수를 만들었는데 이름은 std1이다

== heap 영역에 Student자료형 크기를 선언 할당한다.

static이라는 이름이 들어가 있는 변수는 프로그램 시작 시 바로 static영역에 메모리가 할당된다.
static 변수를 변경하면 해당 변수를 보고 있는 모든 곳에서 같이 변한다
static == 공유 메모리 영역이라고 표현하기도 한다. (정적 메모리 영역)
정적 메모리 영역은 프로그램이 시작 할 때 생성되고 끝날 때 까지 새로 생성되거나 사라지지 않는다.
(heap은 동적 static은 정적이다)

Static

1. 공유 메모리 영역 또는 정적 메모리 영역이라 한다.
    1. **1** 프로그램 시작 시작이 **static**이 붙은 코드들이 모두 **static**영역에 생성되고 **,**프로그램이 종료 될 때 까지 사라지지 않는다**.**
    2. **2 static**영역에 생성된 변수는 어디에서든 공유 할 수 있다**.**
2. 사용 방법은 **:** 클래스명 **.**변수명 **(Student.schoolName** 등등**)**

생성자(Constructor)
() 괄호가 있으면 새롭게 클래스를 만드는 의미이다.

오늘 배운 것

스캐너의 동작 원리

- next와 nextLine의 차이 점;
- 해결 방법
  - Line을 제외한 모든 스캐너 입력 구문에서 입력을 마무리하고 nextLine("")으로 엔터 입력을 지워버린다.
- 입력 받은 값은 buffer에 위치해 있다가 Enter 혹은 Space를 만나고 종료된다
  - 하지만 nextLine은 Space 또한 입력 받기 때문에 Enter로만 종료된다

하나의

06_OOP.pdf 2 MB

객체지향 언어

클래스를 new해서 객체화 하는 것을 = 인스턴스화 한다

Class

클래스 선언부.

[접근제한자][예약어] class 클래스명{};

[대괄호] 는 필요에 따라 쓰거나 안쓰거나 한다.

Class 의 [접근제한자] 의 종류

+ public : 프로젝트 안에서 모두 접근 가능

public class 클래스명 { .....}

~(default):같은 패키지 안에서 접근 가능

class 클래스명 { .....}

필드

[접근제한자] [예약어] class 클래스명 {

필드의 구성 : [접근제한자] [예약어] 자료형 변수명 [= 초기값]

}

+ public : 프로젝트 안에서 모두 접근 가능

protected :해당 클래스 내부 / 같은 패키지 소속 / 상속받은 클래스 에서 접근 가능

~ (default) : 해당 클래스 내부 /같은 패키지 소속

- private :해당 클래스 내부에서만 (getter /setter 사용해야됨)

[예약어]

Static

같은 타입의 여러 객체가 공유할 목적의 필드에 사용하며,

프로그램 start시에 정적 메모리(static) 영역에 자동 할당되는 멤버에 적용

final

하나의 값만 계속 저장해야 하는 변수에 사용하는 예약어 상수