Freaky Bug Public Class #00

Index

LoadMap

2023년 1학기

- 자바
  - 자바 기초
  - 문자열 처리
  - 객체지향 프로그래밍 기초
  - 고급 객체지향 프로그래밍
  - 예외 처리
  - ~~자바 입출력(I/O) 및 파일 처리~~

- 컴퓨터 과학
  - 컴퓨터 내부의 언어체계
  - 전자 회로의 조합논리
  - 메모리와 디스크의 핵심
  - 컴퓨터 내부구조

2023년 2학기

- 자바
    - 자바 입출력(I/O) 및 파일 처리
    - 자바 컬랙션 프레임워크
    - 멀티스레딩 및 동시성
    - 네트워크 프로그래밍
    - 데이터 베이스 연동 (JDBC)
    - ~~고급 자바 주제~~

- 컴퓨터 과학
  - 입출력과 네트워킹
  - 데이터 구조와 처리
  - 프로그래밍 언어처리
  - ~~웹브라우저~~

2024년 1학기

- None

2024년 2학기

- None

Basic Of Java

변수와 자료형

변수란?
많은 사람들이 하는 이야기를 들어보면 변수를 단순히 데이터를 담는 상자 정도로 설명을 하지만,
명확하게 이야기 하자면 변수란 데이터를 조작하기 위한 기본적인 단계 라고 이해하는것이 옳다.

즉 Data Handling 을 위한 기초적인 단계 인 것이다.

자바에서의 변수에 저장되는 자료형에는

1. 기본 자료형 (Primitive Data Type)
2. 참조 자료형(Reference Data Type)

이 있다.

기본 자료형 (Primitive Data Type)

정수형 데이터 타입

```
byte : 8bit = 1byte : -128 ~ 127

short : 16bit = 2byte : -32,768 ~ 32,767

int : 32bit = 4byte : -2,147,483,648 ~ 2,147,483,647

long : 64bit = 8byte : -9,223,372,036,854,775,808 ~ 9,223,372,036,854,775,807
```

실수형 데이터 타입

```
float : 32bit = 4byte : 1.4E-45 ~ 3.4028235E38, 음수 포함

double : 64bit = 8byte : 4.9E-324 ~ 1.79769313486231 57E308, 음수 포함
```

문자형 데이터 타입

```
char (character) : 16 비트크기의 유니코드 문자를 저장할 수 있다.
string : 16 비트크기의 유니코드 문자열을 저장할 수 있다.
```

논리형 데이터 타입

```
boolean : 옳은가? 틀렸는가? true/false 1/0
```

참조 자료형(Reference Data Type)

클래스 : 객체의 참조(메모리 주소)를 저장한다.

사용자 정의 클래스, 인터페이스, 배열

- 사용자가 정의한 클래스 + 자바에서 제공하는 클래스
- 인터페이스 : 사용자가 정의한 자료형
- 배열 : [1, 2, 3, 4, 5]

자료의 형의 변환

암시적 변환 : 자바가 자동으로 수행하는 형 변환
작은것에서 큰것으로 의 변환

명시적 변환 : 사용자가 수동으로 수행하는 형 변환
큰것에서 작은것으로 의 변환

연산자.

자바에서의 연산자와 표현식

변수와 자료형
이를 사용하면 다양한연산자와 표식을 사용해서 값을 계산하고 조작할 수 있다.

산술 연산자 : 두 개의 피연산자 사이에서 산술연산을 수행한다.

```
+ : 덧셈
- : 뺄셈
* : 곱셈
/ : 나눗셈
% : 나머지
```

비교 연산자 : 두 피연사를 비교하여 결과를 논리 값으로 반환한다.

- 논리값 : true, false

```
== : 같다
!= : 같지않다
< : 작다
> : 크다
<= : 작거나 같음
>= : 크거나 같음
```

논리 연산자 : 두 논리 값을 조합하고 결과를 논리값으로 반환한다.

```
&& : AND (논리곱)
|| : OR (논리합)
! : NOT (논리 부정)
```

- 조건문, 반복문에서 주로 사용이됨

비트 연산자 : 비트 단위로 연산을 수행한다

```
& : AND
| : OR
^ : NOT
<< : 왼쪽 시프트
>> : 오른쪽 시프트
```

대입 연산자 : 변수에 값을 할당한다.

Assignment Operator

```
= : 대입
+= : 덧셈 후 대입
-= : 뺄셈 후 대입
*= : 곱셈 후 대입
/= : 나눗셈 후 대입
%= : 나머지 후 대입
<<= : 왼쪽 시프트 후 대입
>>= : 오른쪽 시프트 후 대입
```

```
int a = 10;
int b = 20;

b += a;
```

삼항 연산자 : 조건식을 사용하여 두 개의 값중 하나를 선택한다.

조건식 ? 참 : 거짓

```
a > b ? a - 10 : a + 10
```

조건문과 반복문

프로그램의 흐름을 제어하는데 사용되는 구조이다.

특정 조건에 따라 다른작업을 수행하거나, 동일한 작업을 여러번 반복하는데 사용된다.

조건문 (if문):

```
if문 (만약 ~ 라면) :
주어진 조건이 참일대 실행되는 코드 블록을 포함한다.

if (조건) {
        // 조건이 참일 경우 실행되는 코드
}
```

if-else문 :

주어진 조건이 참일 경우에 실행되는 코드 블록과, 그렇지 않을 경우 실행되는 코드 블록을 포함한다.

```
if (조건) {
        // 조건이 참일 경우 실행되는 코드
} else {
        // 그렇지 않을 경우 실행되는 코드
}
```

if-else if-else문 (else-if문) :

여러 조건을 체크하고, 해당 조건에 따라 실행되는 코드 블록을 포함한다.

```
if (조건1) {
        // 조건1이 참일 경우 실행되는 코드
} else if (조건2){
        // 조건2가 참일 경우 실행되는 코드
} else {
        // 모든조건이 거짓일 경우 실행되는 코드
}
```

```
switch (변수) {
        case 값1 :
                // 변수가 값1과 일치할 경우 실행되는 코드
                break;
        case 값2 :
                // 변수가 값2과 일치할 경우 실행되는 코드
        case 값3 :
                // 변수가 값3과 일치할 경우 실행되는 코드
                break;
        // ...
        default :
                // 변수가 어떤 값과도 일치하지 않을 경우 실행되는 코드
                break;

}
```

반복문 :

for문 : 초기화, 조건, 증감식을 사용하여 코드 블록을 일정 횟수만큼 반복 실행한다.

```
for (초기화; 조건; 증감식) {
        // 조건이 참인동안 반복되는 코드
}
```

while문 : 주어진 조건이 참인 동안 코드 블록을 반복 실행한다.

```
while (조건) {
        // 조건이 참인 동안 반복되는 코드
}
```

do-while문 :
코드 블록을 실행한 후에 조건을 확인하고, 조건 참인동안 코드 블록을 반복 실행 한다.
따라서 코드 블록이 최소한 한 번은 실행된다.

```
do {
        // 조건이 참인동안 반복되는 코드
} while (조건)
```

배열(Array)

배열(Array)은 java에서 동일한 데이터 타입의 여러 데이터를
연속적으로 저장할 수 있는 자료구조이다.
배열은 고정된 크기를 가지며, 한 번 생성된 배열의 크기를 변경할 수 없다.
배열의 각 요소는 인덱스를 사용하여 접근할 수 있으며, 인덱스는 0부터 시작한다.

배열 선언 및 생성:

배열 선언 : 배열을 선언하려면 자료형 뒤에 대괄호 '[]' 를 추가하여 배열임을 나타내야 한다.

```
int[] intArray;
String[] strArray;
```

배열 생성 : new 키워드를 사용하여 크기를 지정하고 배열을 생성한다.

```
intArray = new int[배열의 크기];
strArray = new String[배열의 크기];
```

배열 선언과 동시에 생성을 할 수 있다.

```
int[] intArray = new int[배열의 크기];
String[] strArray = new String[배열의 크기];
```

배열 생성 시 초기값을 지정.

```
int[] intArray = {1, 2, 3, 4, 5};
String[] strArray = {"Apple", "Banana", "Cherry"};
```

배열의 인덱스를 사용하여 개별요소에 접근하여 값을 설정하거나 가져올 수 있다.

```
intArray[0] = 42;
int value = intArray[0];
```

배열의 길이.

배열의이름.length 를 사용하여 얻을 수 있다.

```
Arrayname.length();
```

다차원 배열

java 에서는 다차원 배열을 사용할 수 있으며, 가장 일반적인 형태는 2차원 배열이다.

2차원 배열은 행렬처럼 행(row)과 열(column)을 가진 배열이다.
2차원 배열은 배열의 배열로 생각할 수 있다.
각각의 행이 배열로 구성되어 있다.

2차원 배열의 선언 및 생성 :

2차원 배열은 선언하는 방법은 두 개의 대괄호 '[][]'를 추가한다.

```
int[][] intMatrix;
int[][] strMatrix;
```

new 키워드를 사용하여 배열을 생성할 수 있다.

keyword : 예약어

```
intMatrix = new int[3][4];
[0, 0, 0, 0]
[0, 0, 0, 0]
[0, 0, 0, 0]

strMatrix = new String[3][4];
[null, null, null, null]
[null, null, null, null]
[null, null, null, null]
```

2차원 배열 생성 시 초기값을 지정할 수 있다.

```
int[][] intMatrix1 = {
  {1, 2, 3},
  {4, 5, 6},
  {7, 8, 9}
}

String[][] strMatrix1 = {
  {"Apple", "Banana"},
  {"Cherry", "Orange"}
}
```

배열과 반복문:

배열과 반복문은 자주 함께 사용된다. for문과 for-each문은 배열의 모든 요소에 대해 작업을 수행하는 데 사용할 수 있다.

for문을 사용하여 배열의 모든 요소에 대해 작업을 수행한다.

```
int[] intArray = {1, 2, 3, 4, 5};

for (int i = 0; i < intArray.length; i++) {
    System.out.println(intArray[i]);
}
```

for-each문을 사용하여 배열의 모든 요소에 대해 작업을 수행힌다.

```
int[] intArray = {1, 2, 3, 4, 5};

for (int value : intArray) {
    System.out.println(value);
}
```

2차원 배열과 반복문:

2차원 배열과 반복문을 사용하여 모든 요소에 대해 작업을 수행할 수 있다. 이때, 중첩된 반복문이 사용된다.

for문을 사용하여 2차원 배열의 모든 요소에 대해 작업을 수행한다.

```
int[][] intMatrix = {
    {1, 2, 3},
    {4, 5, 6},
    {7, 8, 9}
};

for (int i = 0; i < intMatrix.length; i++) {
    for (int j = 0; j < intMatrix[i].length; j++) {
        System.out.println(intMatrix[i][j]);
    }
}
```

for-each문을 사용하여 2차원 배열의 모든 요소에 대해 작업을 수행한다.

```
int[][] intMatrix = {
    {1, 2, 3},
    {4, 5, 6},
    {7, 8, 9}
};

for (int[] row : intMatrix) {
    for (int value : row) {
        System..println(value);
    }
}
```

StringHandling

Handling : 조작

"String" 클래스는 불변(Immutable) 객체로, 문자열을 표현하고 처리하는데 사용된다.
문자열을 조작할 때마다 새로운 String 객체가 생성되므로, 반복적인 문자열 조작 작업에는 비효율적일 수 있다.

기본적인 String 메소드

1. length(): 문자열의 길이를 반환.
2. charAt(int index): 주어진 인덱스에 있는 문자를 반환.
3. substring(int beginIndex, int endIndex): 문자열에서 시작 인덱스부터 종료 인덱스 전까지의 부분 문자열을 반환.
4. concat(String str): 주어진 문자열을 원래 문자열에 연결하여 새로운 문자열을 반환.
5. indexOf(String str): 주어진 문자열이 처음 나타나는 위치의 인덱스를 반환, 없으면 -1을 반환.
6. lastIndexOf(String str): 주어진 문자열이 마지막으로 나타나는 위치의 인덱스를 반환, 없으면 -1을 반환.
7. replace(CharSequence target, CharSequence replacement): 문자열에서 대상 문자열을 모두 찾아 교체 문자열로 바꾼 새로운 문자열을 반환.
8. toLowerCase(): 문자열의 모든 문자를 소문자로 변환한 새로운 문자열을 반환.
9. toUpperCase(): 문자열의 모든 문자를 대문자로 변환한 새로운 문자열을 반환.
10. trim(): 문자열의 시작과 끝에서 공백 문자를 제거한 새로운 문자열을 반환합.
11. split(String regex): 주어진 정규식을 기준으로 문자열을 나누어 문자열 배열로 반환.

StringBuilder 클래스와 StringBuffer 클래스

이 두 클래스는 가변(mutable) 객체로, 문자열을 변경하거나 조작할때 사용되며, 효율적인 문자열 처리가 가능하다.

StringBuilder 클래스는 스레드에 안전하지 않은 반변, StringBuffer는 스레드에 안전하는 차이점이 있다.
일반적으로 StringBuiler를 사용하는 것이 성능상 이점이 있다.

주요 메소드

```
append() : 문자열, 숫자, 문자 등 다양한 타입의 값을 빌더에 추가.
insert() : 주어딘 인데스에 무자열, 숫자, 문자 들 다양한 차입의 값을 빌더에 삽입.
delete() : 주어진 시작 인덱스와 끝 인덱스 사이의 문자를 빌더에서 삭제.
replace() : 주어진 시작 인데스와 끝 인덱스 사이의 문자열을 주어진 문자열로 대체.
reverse() : 빌더의 문자열을 반전한다.
length() : 빌더의 문자열을 반환.
setLength(int length) : 빌더의 문자열의 길이를 변경. 길이가 늘어나면 NULL 문자가 채워지며, 길이가 줄어들면 문자가 잘림.
charAt(int index) : 주어진 인덱스에 있는 문자를 반환.
setCharAt(int index, char ch) : 주어진 인덱스에 있는 문자를 새로운 문자로 교체.
toString() : 빌더의 문자열을 'String' 객체로 반환.
```

정규식을 이용한 문자열 처리

정규식 Regular Ecpression

Pattern 클래스와 Matcher 클래스를 사용하여 규식을 이용한 문자열 처리를 수행할 수 있다.

```
import java.util.regex.Pattern;
import java.util.regex.Matcher;

public class RegexExample {
    public static void main(String[] args) {
        String text = "Hello, my email is john.doe@example.com and my friend's email is jane.doe@example.com";
        String emailRegex = "\\b[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\\.[A-Z|a-z]{2,}\\b";

        Pattern pattern = Pattern.compile(emailRegex);
        Matcher matcher = pattern.matcher(text);

        while (matcher.find()) {
            System.out.println("Found email: " + matcher.group());
        }
    }
}
```