3주차 과제

study 할래?

스터디 내용

1. 산술 연산자
2. 비트 연산자
3. 관계 연산자
4. 논리 연산자
5. instanceof
6. assignment (=) operator
7. 화살표 (→) 연산자
8. 3항 연산자
9. 연산자 우선 순위
10. Java 13. switch 연산자

1. 산술연산자

산술 연산자는 덧셈, 뺄셈 등 산술 연산을 수행하는 연산자 (이항 연산자)
부호 연산자는 부호를 지정하는 것들이다. (단항 연산자)

기호	기능
+	덧셈/부호
-	뺄셈/부호
*	곱셈
/	나눗셈
%	나머지

연산의 결과를 변수에 저장할 때는 결과 값이 그 변수의 허용 범위에 들어가는지 여부를 잘 판단해야 한다.
% 연산자의 피연산자는 정수형과 실수형 모두 사용 가능하다.

```
double da = 5.2, db = 4.1;
System.out.println((da%db)); //1.1000000000000005
```

또한 산술연산을 할 때 다음과 같이 자동 형 변환이 일어난다.
두 피연산자 중 하나라도 double 형이면 다른 하나도 double로 변환하고 결과도 double. -> 그렇지 않고 하나라도 float 형이면 다른 하나도 float 결과도 float -> long -> (last) int

2. 비트연산자

비트 연산자는 비트(bit) 단위로 논리 연산을 할 때 사용하는 연산자다. 또한, 비트 단위로 전체 비트를 왼쪽이나 오른쪽으로 이동시킬 때도 사용합니다.

http://www.tcpschool.com/c/c_operator_bitwise

연산자	기능
& (비트 AND)	두 피연산자의 대응되는 비트가 모두 1이면 1을 반환
| (비트 OR)	두 피연산자의 대응되는 비트에서 둘 중 하나가 1이거나 모두 1인 경우 1을 반환
^ (비트 XOR)	두 피연산자의 대응되는 비트에서 서로 같은 경우에는 0을, 다른 경우에는 1을 반환
~ (비트 NOT)	피연산자의 비트를 뒤집음
<<	a의 2진수 표현을 b 비트만큼 왼쪽으로 이동함. 오른쪽은 0으로 채움
>>	a의 2진수 표현을 b 비트만큼 오른쪽으로 이동함. 오른쪽 남는 비트는 버림
>>>	a의 2진수 표현을 b 비트만큼 오른쪽으로 이동함. 오른쪽 남는 비트는 버리고, 왼쪽은 0으로 채움.

3. 관계연산자

연산자	연산자의 기능
<	예) n1 < n2 n1이 n2보다 작은가?
>	예) n1 > n2 n1이 n2보다 큰가?
<=	예) n1 <= n2 n1이 n2보다 같거나 작은가?
>=	예) n1 >= n2 n1이 n2보다 같거나 큰가?
==	예) n1 == n2 n1과 n2가 같은가?
!=	예) n1 != n2 n1과 n2가 다른가?

4. 논리연산자

연산자	연산자의 기능
&&	예) A && B A와 B 모두 true이면 연산결과는 true (논리 AND)
||	예) A || B A와 B 둘 중 하나라도 true이면 연산결과는 true (논리 OR)
!	예) !A 연산결과는 A가 true이면 false, A가 false이면 true (논리 NOT)

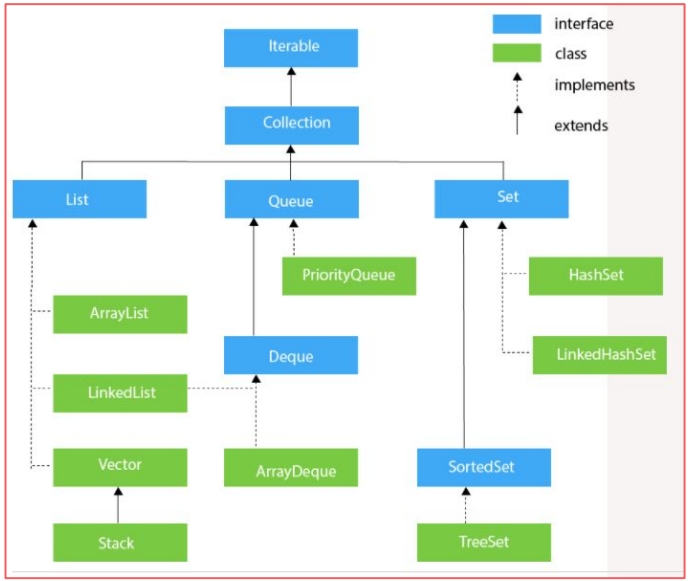
5. instanceof

instaceof는 객체타입을 확인하는데 사용한다.
속성은 연산자이고 형변환 가능 여부를 T/F로 나타낸다.
+추가로 현재 참조하는 클래스를 확인하는 getClass 메소스도 알아두면 좋다. https://improver.tistory.com/139

```
System.out.println(list instanceof List);  //T
//System.out.println(list instanceof LinkedList); //Error
System.out.println(list instanceof Queue); //F
System.out.println(list instanceof Set); //F
//System.out.println(list instanceof HashSet); //Error
System.out.println(list instanceof Collection); //T

System.out.println(hash instanceof Object); //T
System.out.println(hash instanceof Set); //T
System.out.println(hash instanceof List); //F
//System.out.println(hash instanceof ArrayList); //Error
System.out.println(hash instanceof SortedSet); //F

System.out.println(list.getClass()); //class java.util.ArrayList
System.out.println(hash.getClass()); //class java.util.HashSet
```



6.assignment (=) operator

Assignment Operator(대입연산자) : 변수에 값을 대입할때 사용하는 이항 연산자

대입 연산자	설명
=	왼쪽의 피연산자에 오른쪽의 피연산자를 대입함.
+=	왼쪽의 피연산자에 오른쪽의 피연산자를 더한 후, 그 결괏값을 왼쪽의 피연산자에 대입함.
-=	왼쪽의 피연산자에서 오른쪽의 피연산자를 뺀 후, 그 결괏값을 왼쪽의 피연산자에 대입함.
*=	왼쪽의 피연산자에 오른쪽의 피연산자를 곱한 후, 그 결괏값을 왼쪽의 피연산자에 대입함.
/=	왼쪽의 피연산자를 오른쪽의 피연산자로 나눈 후, 그 결괏값을 왼쪽의 피연산자에 대입함.
%=	왼쪽의 피연산자를 오른쪽의 피연산자로 나눈 후, 그 나머지를 왼쪽의 피연산자에 대입함.
&=	왼쪽의 피연산자를 오른쪽의 피연산자와 비트 AND 연산한 후, 그 결괏값을 왼쪽의 피연산자에 대입함.
|=	왼쪽의 피연산자를 오른쪽의 피연산자와 비트 OR 연산한 후, 그 결괏값을 왼쪽의 피연산자에 대입함.
^=	왼쪽의 피연산자를 오른쪽의 피연산자와 비트 XOR 연산한 후, 그 결괏값을 왼쪽의 피연산자에 대입함.
<<=	왼쪽의 피연산자를 오른쪽의 피연산자만큼 왼쪽 시프트한 후, 그 결괏값을 왼쪽의 피연산자에 대입함.
>>=	왼쪽의 피연산자를 오른쪽의 피연산자만큼 부호를 유지하며 오른쪽 시프트한 후, 그 결괏값을 왼쪽의 피연산자에 대입함.
>>>=	왼쪽의 피연산자를 오른쪽의 피연산자만큼 부호에 상관없이 오른쪽 시프트한 후, 그 결괏값을 왼쪽의 피연산자에 대입함.

6-1. assignment (=) operator

&=	왼쪽의 피연산자를 오른쪽의 피연산자와 비트 AND 연산한 후, 그 결괏값을 왼쪽의 피연산자에 대입함.
|=	왼쪽의 피연산자를 오른쪽의 피연산자와 비트 OR 연산한 후, 그 결괏값을 왼쪽의 피연산자에 대입함.
^=	왼쪽의 피연산자를 오른쪽의 피연산자와 비트 XOR 연산한 후, 그 결괏값을 왼쪽의 피연산자에 대입함.

```
6  int a = 30 ;
7  int b = 11 ;
8  //30을 2진법으로 바꾸면 11110(2), 11 -> 2진법 = 1011(2)
9  a &= b;
10 // &= 는 2진법으로 교체 후 자리수를 비교해 같은 숫자를 조합한다. 01010(2)
11 System.out.println("a &= b : " + a);  //10
12 a = 30; b = 11;
13 a |= b;
14 //|= 는 2진법으로 교체 후 1이 있는 자리수를 조합한다. 11111(2) = MAX
15 System.out.println("a |= b : " + a);  //31
16 a = 30; b = 11;
17 a ^= b;
18 //^= 는 2진법으로 교체 후 자리수를 비교해 다른 숫자의 자리에 1을 넣어 조합한다. 10101(2)
19 System.out.println("a ^= b : " + a);  //21
20 }
```

2진법으로 피연산자들을
변환한 뒤 AND, OR, XOR으로
계산해 대입하는 연산자

6-2. assignment (=) operator

<<=	왼쪽의 피연산자를 오른쪽의 피연산자만큼 왼쪽 시프트한 후, 그 결괏값을 왼쪽의 피연산자에 대입함.
>>=	왼쪽의 피연산자를 오른쪽의 피연산자만큼 부호를 유지하며 오른쪽 시프트한 후, 그 결괏값을 왼쪽의 피연산자에 대입함.
>>>=	왼쪽의 피연산자를 오른쪽의 피연산자만큼 부호에 상관없이 오른쪽 시프트한 후, 그 결괏값을 왼쪽의 피연산자에 대입함.

'A << B' 라는 수식이 있다고 하면, A 라는 녀석의 비트 값을 B만큼 왼쪽으로 이동 시킨다.

쉬프트 연산자 참고 : Empty :: 자바의神 Vol.1 : 비트 시프트 연산자 (tistory.com)

7.화살표(-↠)연산자

→ 연산자는 "식별자없이 실행가능한 함수"로 람다식이라 불린다.
함수인데 함수를 따로 만들지 않고 코드 한 줄에 함수를 써서 호출한다.
람다식 문법 : **(매개변수, ...) -> {실행문...}** ->기호는 매개변수를 이용해서 {}를 실행한다는 뜻.

```
4  //람다식에서는 무조건 functional interface 사용해야함
5  interface Say{ //인터페이스 안에 메서드가 한개밖에 없는경우 functional interface
6      void something(int a, int b);
7  }
```

개념 : https://www.youtube.com/watch?v=foC6t8dZHls

간단예제 : https://coding-factory.tistory.com/265

```java
interface Say{
    void something();
}
class Person{
    public void greeting(Say line){
        line.something();
    }
}

public class Solution {

    public static void main(String[] args) {
        //Lambda Expression
        //Java 8 JDK 1.8
        //여러코드를 메서드 안에 넣는것
        //람다가 없던시절 1. interface
        Person rin = new Person();
        rin.greeting(new Say(){
            @Override
            public void something() {
                System.out.println("my name is Rin.");
            }
        });
        System.out.println("========================");
        //람다 사용 (리턴x/매개인자x)
        rin.greeting( () -> {
            System.out.println("My name is Rin.");
        });
}
}
```

Problems | Javadoc | Declaration | Progress | Console

```
<terminated> Solution [Java Application] C:₩Users₩user₩Desktop₩java₩OOPSW₩jdk1.8.0
my name is Rin.
========================
My name is Rin.
```

```java
interface Say{
    int something();
}
class Person{
    public void greeting(Say line){
        int num =  line.something();
        System.out.println("my number is " + num);

    }
}
public class Solution {

    public static void main(String[] args) {
        //Lambda Expression
        //Java 8 JDK 1.8
        //여러코드를 메서드 안에 넣는것
        //람다가 없던시절 1. interface
        Person rin = new Person();
        rin.greeting(new Say(){
            @Override
            public int something() {
                System.out.println("my name is Rin.");
                return 5;
            }
        });
        System.out.println("========================");
        //람다 사용 (리턴 o/매개인자x)
        rin.greeting( () -> {
            System.out.println("My name is Rin.");
            return 8;
        });
```

Problems | Javadoc | Declaration | Progress | Console

```
<terminated> Solution [Java Application] C:₩Users₩user₩Desktop₩java₩OOPSW₩jdk1.8.0
my name is Rin.
my number is 5
========================
My name is Rin.
my number is 8
```

```java
interface Say{
    int something(int a);
}
class Person{
    public void greeting(Say line){
        int num = line.something(3);
        System.out.println("my number is " + num);

    }
}
public class Solution {

    public static void main(String[] args) {
        //Lambda Expression
        //Java 8 JDK 1.8
        //여러코드를 메서드 안에 넣는것
        //람다가 없던시절 1. interface
        Person rin = new Person();
        rin.greeting(new Say(){
            @Override
            public int something(int a) {
                System.out.println("my name is Rin.");
                System.out.println("parameter value is " + a );
                return 5;
            }
        });
        System.out.println("========================");
        //람다 사용 (리턴 o/매개인자o) 1. 타입을 써주지 않아도됨(선택) 2. 괄호도 없애도됨
        rin.greeting( (int a) -> {
            //rin.greeting( a -> {
            System.out.println("My name is Rin.");
            System.out.println("parameter value is " + a );
            return 8;
        });
```

Problems | Javadoc | Declaration | Progress | Console

```
<terminated> Solution [Java Application] C:₩Users₩user₩Desktop₩java₩OOPSW₩jdk1.8.0_71₩bin₩ja
my name is Rin.
parameter value is 3
my number is 5
========================
My name is Rin.
parameter value is 3
my number is 8
```

```java
interface Say{
    int something(int a, int b);
}
class Person{
    public void greeting(Say line){
        int num = line.something(3, 5);
        System.out.println("my number is " + num);

    }
}
public class Solution {

    public static void main(String[] args) {
        //Lambda Expression
        //Java 8 JDK 1.8
        //여러코드를 메서드 안에 넣는것
        //람다가 없던시절 1. interface
        Person rin = new Person();
        rin.greeting(new Say(){
            @Override
            public int something(int a, int b) {
                System.out.println("my name is Rin.");
                System.out.println("parameter values are " + a +"," + b );
                return 5;
            }
        });
        System.out.println("========================");
        //람다 사용 (리턴 o/매개인자多) 1. 타입을 써주지 않아도됨(선택) 2. 여러개 일때는 괄호 필요
        rin.greeting( (int a, int b) -> {
            //rin.greeting( (a, b) -> {
            System.out.println("My name is Rin.");
            System.out.println("parameter values are " + a +"," + b );
            return 8;
        });
```

Problems | Javadoc | Declaration | Progress | Console

```
<terminated> Solution [Java Application] C:₩Users₩user₩Desktop₩java₩OOPSW₩jdk1.8.0_71₩bin₩javaw.exe (2020.
my name is Rin.
parameter values are 3,5
my number is 5
========================
My name is Rin.
parameter values are 3,5
my number is 8
```

```java
interface Say{
    void something(int a, int b);
}
interface Hello{
    void something(String a, String b);
}
class Person{
    public void greeting(Say line){
        line.something(3, 5);
    }
    //overloading
    public void greeting(Hello line){
        line.something("3", "5");
    }
}
public class Solution {

    public static void main(String[] args) {
        //Lambda Expression
        //Java 8 JDK 1.8
        //여러코드를 메서드 안에 넣는것
        //람다가 없던시절 1. interface
        Person rin = new Person();
        rin.greeting(new Say(){
            @Override
            public void something(int a, int b) {
                System.out.println("parameter values are " + a +"," + b );
            }
        });
        System.out.println("========================");
        //람다 사용 1. 타입을 반드시 써줘야 하는경우
        //rin.greeting( (int a, int b) -> {
            rin.greeting( (a, b) -> {
            Syst                                    +"," + b );
        });
}
```

The method greeting(Say) is ambiguous for the type Person

Problems | Javadoc | Declaration | Progress | Console

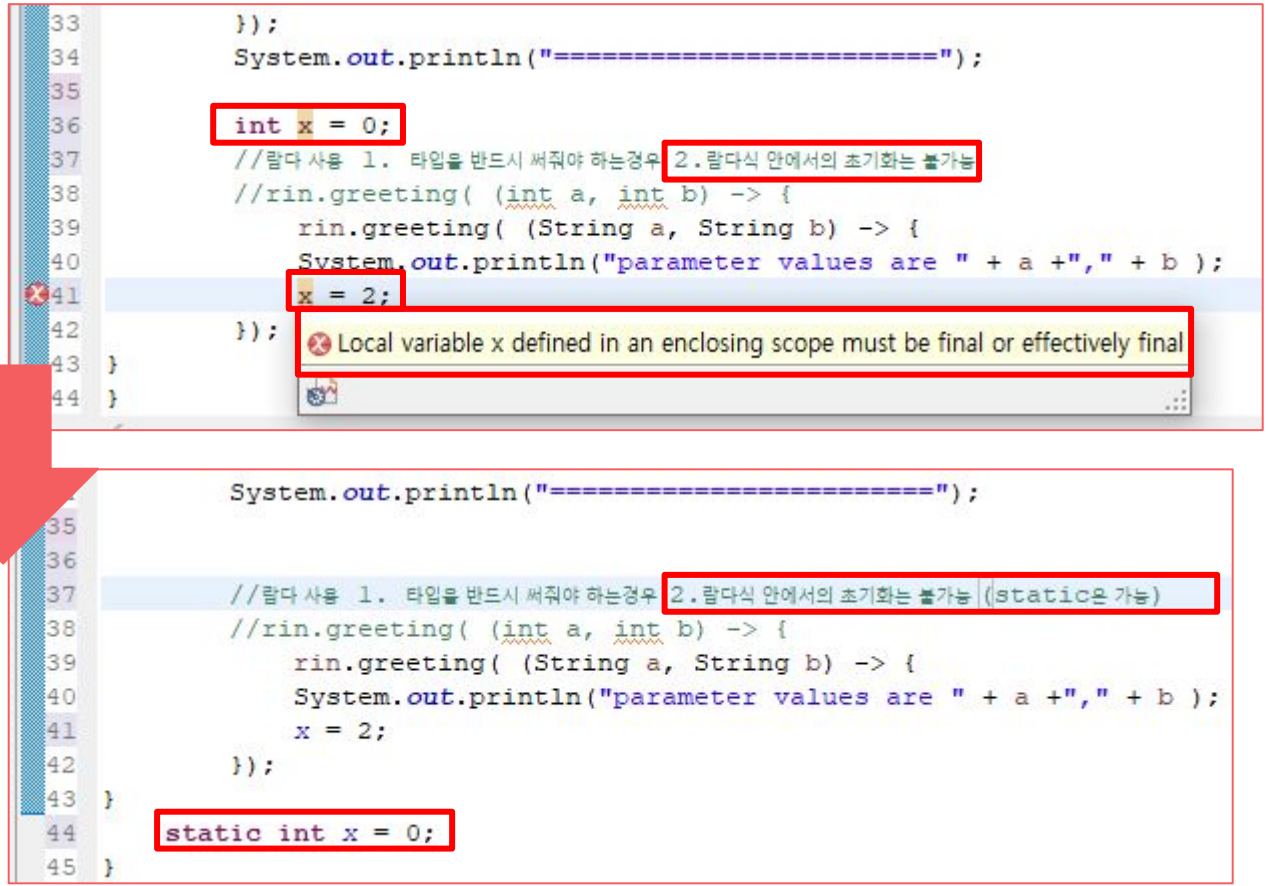
```
<terminated> Solution [Java Application] C:₩Users₩user₩Desktop₩java₩OOPSW₩jdk1.8.0_71₩bin₩javaw.exe (2020.
parameter values are 3,5
========================
parameter values are 3,5
```

```java
interface Say{
    void something(int a, int b);
}
interface Hello{
    void something(String a, String b);
}
class Person{
    public void greeting(Say line){
        line.something(3, 5);
    }
    //overloading
    public void greeting(Hello line){
        line.something("3S", "5S");
    }
}
public class Solution {

    public static void main(String[] args) {
        //Lambda Expression
        //Java 8 JDK 1.8
        //여러코드를 메서드 안에 넣는것
        //람다가 없던시절 1. interface
        Person rin = new Person();
        rin.greeting(new Say(){
            @Override
            public void something(int a, int b) {
                System.out.println("parameter values are " + a +"," + b );
            }
        });
        System.out.println("========================");
        //람다 사용 1. 타입을 반드시 써줘야 하는경우
        //rin.greeting( (int a, int b) -> {
            rin.greeting( (String a, String b) -> {
            System.out.println("parameter values are " + a +"," + b );
        });
}
```

Problems | Javadoc | Declaration | Progress | Console

```
<terminated> Solution [Java Application] C:₩Users₩user₩Desktop₩java₩OOPSW₩jdk1.8.0_71₩bin₩javaw.exe (2020.
parameter values are 3,5
========================
parameter values are 3S,5S
```


});
System.out.println("========================");
int x = 0;
//람다 사용 1. 타입을 반드시 써줘야 하는경우 2.람다식 안에서의 초기화는 불가능
//rin.greeting((int a, int b) -> {
rin.greeting((String a, String b) -> {
System.out.println("parameter values are " + a +"," + b);
x = 2;
});
Local variable x defined in an enclosing scope must be final or effectively final
}
}
System.out.println("========================");
//람다 사용 1. 타입을 반드시 써줘야 하는경우 2.람다식 안에서의 초기화는 불가능 (static은 가능)
//rin.greeting((int a, int b) -> {
rin.greeting((String a, String b) -> {
System.out.println("parameter values are " + a +"," + b);
x = 2;
});
}
static int x = 0;
}

7.화살표(-↠)연산자

람다 (익명함수) 의 장단점

장점

1. 코드의 간결성
2. 지연연산 수행
3. 병렬처리 가능 - 멀티쓰레드를 활용

단점

1. 호출이 까다로움
2. 불필요하게 너무 사용하면 가독성이 떨어짐

8. 3항연산자

3항 연산자는 `(조건식) ? 피연산자1 : 피연산자2;` 로 사용된다. 조건식이 true이면 결과는 피연산자1이고, false이면 결과는 피연산자 2이다.

간단예제)

```
String b = (true) ? "true" : "false" ;
System.out.println(b);  //true

String b1 = "";
if (false) {
    b1 = "true";
}else{
    b1 = "false";
}
System.out.println(b1); //false
```

9. 연산자 우선순위

종 류	연산방향	연산자	우선순위
단항 연산자	←	++ -- + - ~ ! (타입)	높음
산술 연산자	→	* / %	
	→	+ -	
	→	<< >> >>>	
비교 연산자	→	< > <= >= instanceof	
	→	== !=	
논리 연산자	→	&	
	→	^	
	→	|	
	→	&&	
	→	||	
삼항 연산자	→	?:	
대입 연산자	←	= *= /= %= += -= <<= >>= >>>= &= ^= |=	낮음

10. Java13 switch 연산자

이전 Java switch case 의 문제점

1.불필요하게 장황하다　　2.Error 발생시 디버깅 어려움　　3.Missing Break;

"자바 13버전 이전에 12버전에서는 어떻게 해결하려 했을까?"

일단 jdk 12 랑 13이랑 크게 차이나지는 않지만, 먼저 설명을 하면 lambda 연산을 사용할 수 있다.

```
switch (day){
    case "MONDAY", "TWO", "SUNDAY" -> System.out.println(6);
    case "NOT" -> System.out.println("not");
    default -> System.out.println("default");
}
```

Case 문에 람다 X -> Y; 람다식?

이게 "case L ->" 방식으로 문법을 작성할 수 있다. 근데 섞어서 사용하는게 불가능하다. 예를들어

```
switch (day){
    case "MONDAY", "TWO", "SUNDAY" -> System.out.println(6);
    case "NOT" : System.out.println("not");
    default -> System.out.println("default");
}
```

이런 방식으로 람다로 해결하려 했는데, 저런식으로 "-> " 와 " : " 를 혼동하여 사용할 수 없다.

두번째로는 multi case Label이다. 원래 저렇게 Mon, Two, Sun 3개를 해줄 수 없는데, 12 이후로는 저렇게 3가지의 조건을 걸어줄 수 있다는 단점이 있다. 벌써 case 문을 작성할게 확 줄어들었다.

출처 : https://catch-me-java.tistory.com/31

12에서는 switch 문을 통해 리턴값을 반환할 수 있다.

Java Switch var 사용해보기

```
int result = switch (day){
    case "MONDAY" -> 2;
    case "FRIDAY" -> 3;
    case "NOT" -> 43;
    default -> throw new IllegalStateException("Unexpected val
};
System.out.println(temp);
```

이런식으로 switch 리턴값을 반환한다. 어 ? 그럼 var 로도 가능하지 않을까 해서 응용해보면

```
var result : Serializable & Comparable<? extends Serializable & Comparable<?>
    case "MONDAY" -> "2";
    case "FRIDAY" -> 3;
    case "NOT" -> "Not";
    default -> throw new IllegalStateException("Unexpected v
};
```

이런식으로 var 타입을 사용해서 문제를 더 재밌게 접근할 수 있다.

출처 : https://catch-me-java.tistory.com/31

Java 13 이후로 뭐가 추가 되었나요?

자료가 정말 없어서 jdk로만 보다보니 좀 다를 수 있지만 일단 yield 라는 산출값을 리턴이 가능하다. 예전에는 break를 통해서 했는데 yield를 사용하여 변수 에 값을 넣을 수 있다.

```
public static void main(String[] args) {
    String str = "hello";

    int value = switch (str) {
        case "hiHowAreYou":
            System.out.println("I am not just str!");
            yield 1;
        case "hello":
            System.out.println("Me too.");
            yield 2;
        default:
            System.out.println("OK");
            yield -1;
    };
    System.out.println(value);
}
```

이런식으로 가능하다. 나중에 break 와 yield 에 대해서 알아보도록 하자 :D