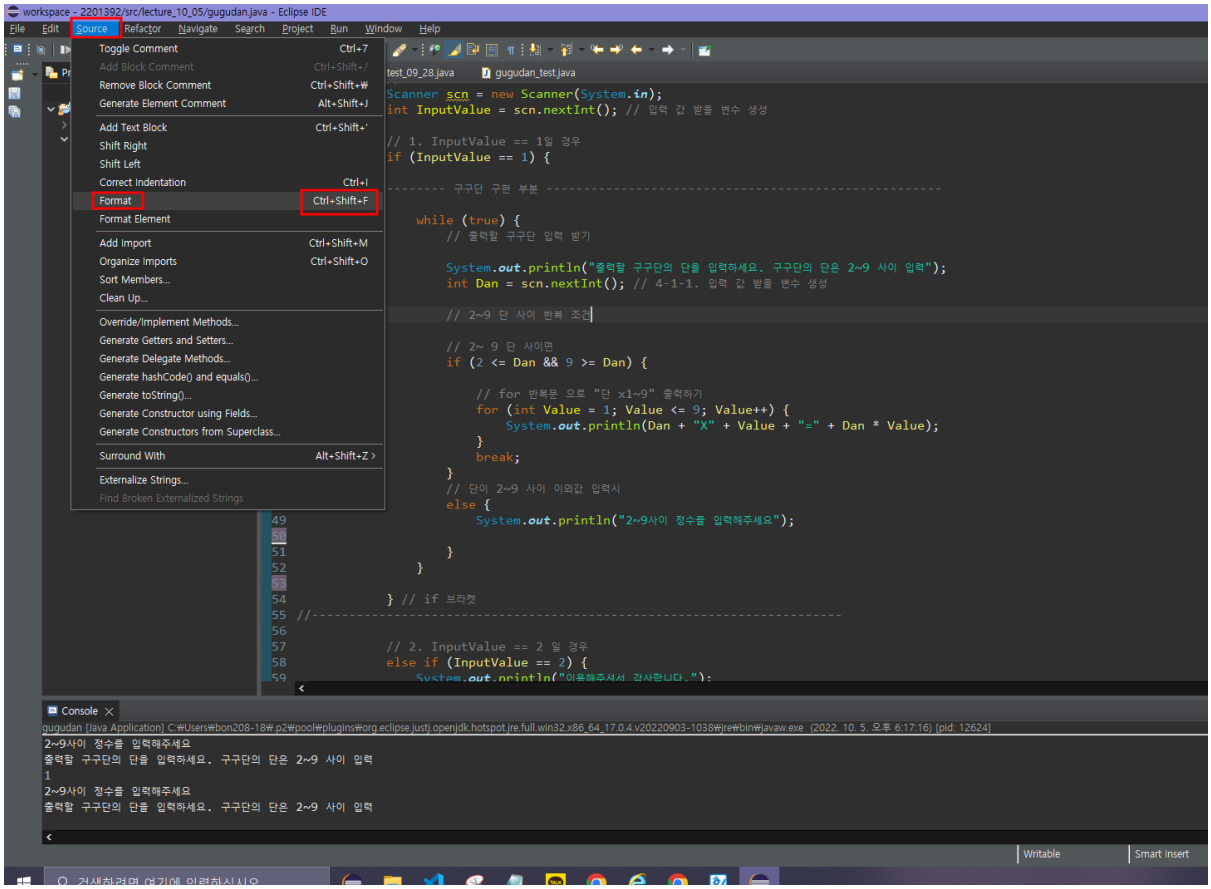
22. 10 .05 - 이항연산자

10. 05 자바수업

ctl + shift + f ⇒ 자동 들여쓰기

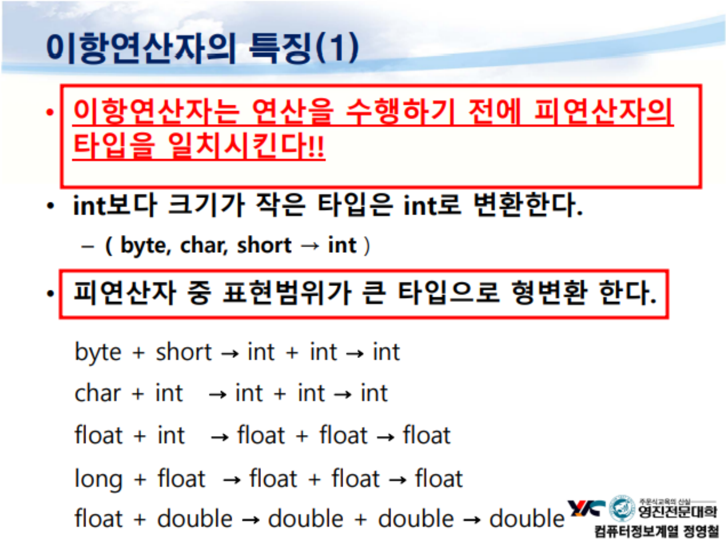


or	and
||	&&

연산자

새로운 언어를 접할 때 연산자를 공부하는 관점

1. 기능별로 나눠서 (산술, 논리 등)
2. 연산자의 우선순위
3. 항의 갯수 (피연산자수)



이항연산자는 연산을 수행하기 전에 피연산자의 타입을 일치 시킨다

이항(二項)연산자 (피연산자 2개 ,좌항과 우항의 값으로 이뤄져 있다)

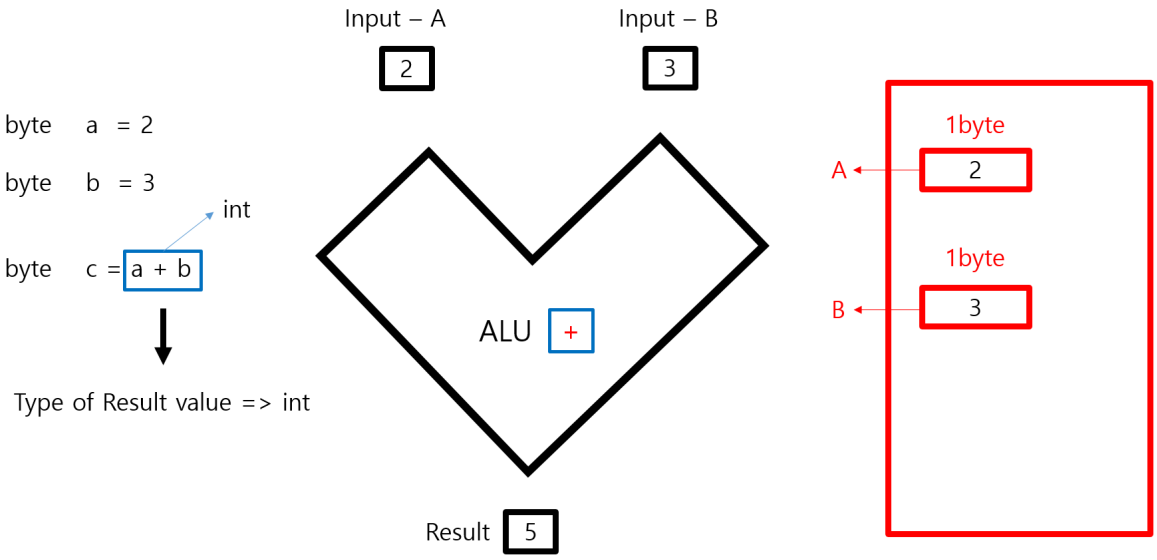
> 💡 **CPU 안에 A**rithmetic **L**ogic **U**nit 이 있다.

※※

- 컴퓨터에서 저장하는 `데이터`는 전부 `2진수`**(0 또는 1)** 로 **저장** 된다.
- **2진수로 저장되는 방식**이 `자료의 형태`에 따라 **틀려진다**.
    - → 컴퓨터 프로그래밍 언어 [`숫자`/ `문자`/ `불린`]
      → `숫자`(정수/ 실수) , `문자`, `불린(논리)`에 따라 **비트 표현 방식이 틀리다**
      → 정수와 실수의 비트 표현 방식도 틀리다
- `자료형`이 다르면 **비트 연산을 할 수 없어서** 좌항과 우항의 자료형을 일치 시켜야 한다.
    - **ALU는 비트 단위로 연산**을 한다 ⇒ **자료형의 비트 표기 방법이 같아야 한다.**

- 때문에 ⇒ **어떤규칙으로 해당 언어에서 이항연산자의 자료형을 일치를 시켜주는지 잘 알아야 한다.**

※※



> 값은 실제로 작더라도(int 이하) , 그 값을 퍼와서 ALU 메모리에 (Input - A , Input - B) 넣어야 한다.
> Input - A , Input - B 도 내부적으로 메모리 공간이 있다(자료형, 크기).

ALU 안 에서 만들어진 메모리 공간의(내부 메모리의 값) 자료형은 3개다 일치 해야한다.

⇒ 피연산자 2개(이항연산자인경우) 의 자료형과 ,
연산을하고 나온 결과 값을 담을 임시 메모리 공간의 자료형이 당연히 같아진다. (같은것 끼리 연산해서)

※ALU는 비트 단위로 연산을 한다 ⇒ 자료형의 비트 표기 방법이 같아야 한다.

■ 정수 자료형에 대한 논의

✓ 자바의 4가지 정수 자료형

- byte, short, int, long
- 정수를 표현하는데 사용되는 바이트 크기에 따라서 구분이 됨

✓ 작은 크기의 정수 저장에는 short? 아니면 int?

- CPU는 int형 데이터의 크기만 연산 가능
- 때문에 연산직전에 short형 데이터는 int형 데이터로 자동변환
- 변환의 과정을 생략할 수 있도록 int를 선택한다!

✓ 그럼 short와 byte는 왜 필요한가?

- 연산보다 데이터의 양이 중요시 되는 상황도 존재!
- MP3 파일, 동영상 파일
- 데이터의 성격이 강하다면 short와 byte를 활용!

> ※ **정수(좌항) 과 실수(우항) 인 경우** → **우선순위가 높은 것으로 일치 시킨다** ※
>
> 정수보다는 **실수가 값의 표현 범위**가 더 크기 때문에 실수가 우선권을 가져서
> 좌항 우항 전부 실수형으로 바꾼다.
> 우선순위가 높은 것으로 일치 시킨다

> ※ **대입 연산** 시 , **좌항에 있는 자료형의 크기가 우항에있는 자료형 보다 작을 경우** ※
>
> 오버플로우나, 언더플로우가 발생할 수 있다.

※ 변수일 때 ※

JVM에서 ALU를 설계 時 , ALU내부에 들어오는 값을 저장할 내부 메모리 공간이 있는데

ALU 내부 메모리 공간도 **자료형**과 **크기**를 **가진다** ⇒ 자료형과 크기도 맞춰야 한다
정수인 경우 2가지 [**Int형 (4 바이트)** / **long형 (8바이트)**] 타입으로 분류 해 놓았다.

때문에 Int 이하의 자료형은 전부 Int형으로 집어 넣는다 ⇒ 크게 만들어 놓으면 문제 없기 때문!!!

> **※ 메모리상에서 데이터를 가져 오면 크기가 있다 ※**
>
> ALU를 설계할 때 ALU에 들어오는 값을 저장할 메모리 공간이 있다.
> ⇒ **정수인 경우 JVM에서 2가지로 설정해 놓았다.**

※ ALU 전처리 과정에서 이항 연산 時 ※

Int 이하의 자료형은 전부 Int형으로 집어 넣는다 ⇒ 크게 만들어 놓으면 문제 없기 때문!!!

1) int 이하의 자료형은 이항 연산시 무조건 -> int 형으로 전환
2) 자료형의 우선순위는 long > int (큰수를 담을 수있는 자료형으로 맞춘다)

> ex) 메모리상의 값은 바이트 + 바이트를 하더라도 ALU에 집어 넣으면 각각의 값을 복사를 해서 집어 넣어야하는데 이때 INT 이하의 자료형은 전부 INT로 컨버팅이 되어 집어 넣어 버린다. ⇒ 결과 값으로 INT형이 나온다.
> 대입연산자에서 우항에 있는 값을 좌항으로 옮기려고 하는데
> 우항은 INT 좌항은 Byte 인경우 **오버플로우나 언더플로우가 발생**할 수 있다.

💡 Q. int 보다 작은 값이 입력되는 경우
1) 이항연사자가 연산 時, 좌항 우항의 우선순의를 먼저 맞추나 ?
2) int 로 먼저 변환을 시켜 버리나?

⇒ JVM에서 **전처리 과정(Pre-Processing) 을 거쳐 값이 입력 되자마자 int형으로 변경해 준다.**
전처리 = 기본 반응이나 가공에 앞서 화학적 · 물리적 작용을 가하여 예비적으로 하는 처리

```
byte a = 5
short b = 3

    1) (Byte)   (short)
long c = a    +    b     => int형으로 전
    2) (int)      (int)

long c = long(a+b)  // 이렇게 해야 오버플로
                    //     로우플러우 안생
```

1.먼저 계산

byte c = (byte) (a + b);

2. byte로 형변환

3. 대입시킨다.

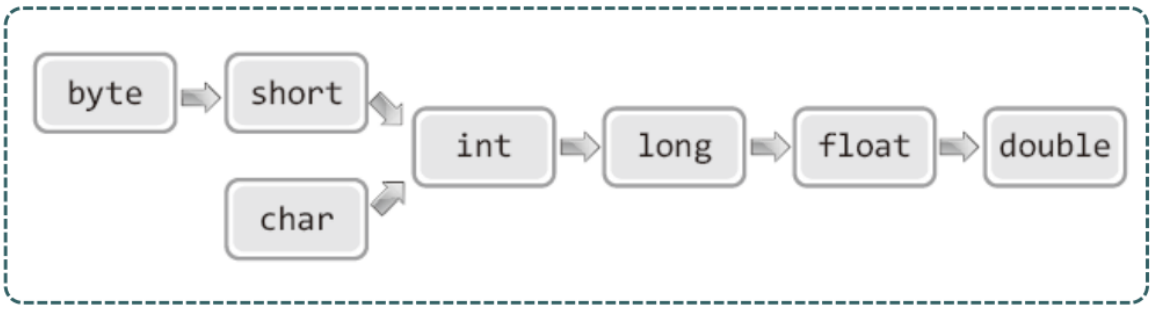
명시적(강제)형 변환 [Type casting]

사용 방법 : (변경하고자 하는 자료형) 피연산자
(int) 3.0f

■ 자동 형 변환(Implict Conversion)

JAVA 공부한 적 있어?

✓ 자동 형 변환 규칙




```
double num2=3.5f+12;
```
12가 12f로 자동 형 변환

문자형 char(캐릭터) 도 결국 **정수**로 저장됨 (2Byte)

char 과 char을 이항연산 한다면 int형이 나온다.

상수가 이항연산 처리가 되면 전처리 과정을 거친다 = > 상수 간에 연산은 문제가 없다

✓ 자바의 문자 표현

- 문자 하나를 2바이트로 표현하는 유니코드 기반으로 표현
- 유니코드는 전 세계의 문자를 표현할 수 있는 코드 집합
- 문자는 작은 따옴표로 표현한다.
- 문자는 char형 변수에 저장한다. 저장 시 실제로는 유니코드 값 저장

오른쪽의 두 코드는

사실상 동일한 코드

```
char ch1='A';
char ch2='한';
```

변환 발생

```
char ch1=65;        // 65는 16진수로 0x41
char ch2=54620;     // 54620은 16진수로 0xD55C
```

8bits -> is used to be presented data into bites

2^8 -> 256 -> 0 ~ 255

8비트중에서 1비트를 +,-를 담고 나머지 7비트가 남는다 -128 ~ 127

-0 , 0+ 두개가 나와서 2^7 -1

byte c = 127 or 126+1 ; 는 전처리 과정에서 byte로 바꿔 버림

상수는 값이 정확하게 예측이 된다. 상수라

byte c = 128 이상 오류 (-128 ~ 127)

char	String
문자 형	문자열
'b '	" betty"

char 문자 형 ⇒ 비트로 저장되는데 정수로 표현한다

String 을 사용

char 를 배열 형식으로 사용

> 💡 **문자**를 비트로 저장하는 것은 "**정수**로 저장한다"

⇒ 유니코드 테이블 보고 해당 문자에 해당하는 정수 값을 이용한다

때문에 일반적인 산술 연산이 가능하다.

자료형의 우선순위는 항상 더 많이 담을 수 있는 숫자를 나타낸다

int 보다 크기가 작은 타입은 int로 변환한다.

JVM ALU 변수를 설정할 때

정수 파트에서 인트, 롱 으로 나눠서 만들어 놨다. (4바이트 8바이트)

인트 이하의 자료형은 전부 인트형으로 집어 넣는다

때문에 결과 값은 인트 이상의 값이 무조건 나온다 .

byte a = 10;
byte b = 20;
byte c = a + b;

```
byte a = 10;  // ALU 메모리 공간에 들어가면 Int 로 바뀐다.
byte b = 20;  // ALU 메모리 공간에 들어가면 Int 로 바뀐다.
byte c = a + b;    // 이항 연산을 하게 되면 int + int 로 바뀐다
                      =>

System.out.println( c );
```

```
byte a = 2
byte b = 3
byte c = a+b // 출력이 되지 않고 c는 오류가 난다

- (1) 오류가 나는 이유
-> a+b 는 int이지만 c는 byte이기 때문에
-> 자료형이 int와 같거나 int보다 높은 long이어야 한다.

- (2) a+b가 byte가 아니고 int인 이유
-> 자료형이 int 이하이면 그 자료형을 int로 변환

- (3) int 이하이면 자료형을 int로 변환하는 이유
-> JVM에서 ALU 설정 시 4byte와 8byte만 취급하기 때문에
-> int 이하인 자료형은 다 int로 취급
```

```
// 명시적(강제)형 변환 -> type casting
byte a = 2
byte b = 3

byte c = (byte)(a+b)


// 자료형 숫자의 범위
// 8bits  2^8 -> 256 -> 0 ~ 255
// -128 ~ 127

byte = 128;   -> 오류
byte = 126+2; -> 오류
byte = 127;   -> 오류가 나지 않음
```

```
// (1) 변수, 정수형에서 이항 연산을 처리하는 규칙
//   1) int 이하의 자료형은 무조건 -> int 형으로 변환
//   2) 자료형의 우선 순위는 long > int
byte a = 2;
byte b = 3;
int c = (int)(a+b);
-------------------------------------------
// (2) 변수, 실수
//    1) 자료형의 우선 순위 double > float
//       -> 자바의 실수 타입의 기본 처리는 double
float  a = 3.0f; // f를 사용하는 이유 double과 구분
double b = 4.0;
float c = a+b;
* 오류 나는 이유
  -> float보다 double의 바이트가 더 높기 때문에 오류 발생
-------------------------------------------
// (3) 변수, 정수 <-> 실수
//   1) 실수 > 정수
//     -> 수의 범위는 지수
float a = 3.0f;
int   b = 4;
double c = a + b;
  -> a는 float이고 b는 int이다 자료형이 다를 시  수의 범위가 더 높은 것으로 변환
  -> b는 int 정수는 실수보다 작기 때문에 b의 자료형은 float가 됨
  -> double은 float 보다 크기가 크기 때문에 출력 가능
```

```
// 문자 vs 문자열
// 문자: 1글자        -> 'a' 문자
// 문자열: 1글자 이상  -> "a" 문자열
// char(정수) -> 문자형 / 아스키코드
// 문자열     -> String, char array [배열 형식]
-----------------------
char a = 97; -> a 출력
char b = 2;
char c = a+b;
-----------------------
char a = 97;
a++; -> b 출력
-----------------------
String bar = "11";

system.out.println(bar + 2.3); // 112.3 출력
-> 자료형이 다를 시 문자열로 변환해서 붙인다.
-> 문자열이 우선 순위가 가장 빠르다.
-> /,*,- 는 불가 하지만 자바스크립트는 가능
```

and 이항

not 단항

より強いのほうが勝つ

문자열> 숫자　必ず文字列が優先

자바에서는 유니코드를 사용하기 때문에 char를 2바이트로 사용한다.

변수는 크게 2 종류

primitive variable

reference variable ⇒ (String)