NCS - Java
002 Basic Syntax
(기본적인 문법)

1. 코드 작성 규칙 (Coding Rules)

1. Java 는 기본적으로 C와 C++의 기본적인 문법들을 계승하였기 때문에 C를 접해본 개발자는 더 쉽게 이해할 수 있다

; 세미콜론

문장의 끝에는 ; (세미콜론)을 붙여야 한다
단, 블럭의 시작줄인 { (오픈 브라켓) 다음에는 붙이지 않는다

```
System.out.println("안녕 Java");
```

{ } 블럭

블럭은 묶여진 하나의 영역(Scope)으로 내부에 선언된 변수나 함수(Method) 들은 동일한 영역에서 실행되는 것으로 간주된다

```
public static void main(String args[]) {
   ... ;
}
```

칸 띄우기

연속된 Space (빈 공간)는 하나의 공간으로 인식된다
* Java에서 New Line(줄 띄움)은 아무런 문법적 구속력이 없다

```
public      static      void      main(String      args[]) {
   ... ;
}
```


2. 주석 (Comments)

1. 주석 처리 또한 들여쓰기와 함께 개발자에게 요구되는 필수적인 코드 작성규칙이라고 할 수 있다.

//	한 줄을 주석처리한다

```
// System.out.println("안녕 Java");
```

/* */	여러줄을 블록으로 주석처리 한다

```
/ * public static void main(String args[]) {
   ... ;
} */
```

/** */	Java Document 를 만들 때 사용한다. javadoc 명령어와 함께 사용되며, javadoc 파일명.java 라고 하면 자동으로 문서를 생성해준다

```
/**
 * @param 정수형 입력값을 받는다
 */
```


3. 식별자 (Identifiers)

1. 개발자가 임의로 만드는 클래스, 함수(Method), 변수(Variable, Constant)의 이름을 말한다

대소문자 구분

대소문자를 구분하고, 식별자의 길이에는 제한이 없지만
너무 긴 길이는 가독성을 떨어뜨릴 수 있다

```
public class HelloWord {
    String firstName = "Michael";
}
```

첫 글자에 숫자는 안됨

식별자에 영문자, 한글, 숫자, 밑줄 _ , 달러 $ 를 사용할 수 있다.
단, 숫자로 시작할 수는 없다

```
int firstName$1 = "Michael";   // 사용 가능
int 1stName     = "Michael";   // 사용 불가
```

키워드 사용불가

Java 역시 미리 정의되어 있는 예약어(Keywords)는 사용할 수 없다

int, long, public, try, new 등의 예약어는 식별자로 사용될 수 없다

4. 예약어 (Keywords)

1. 아래 예약어들은 식별자(이름)로 사용될 수 없다

분 류	키 워 드
기본 자료형	boolean, byte, char, short, int, long, float, double
접근 지정자	private, protected, public
클래스 관련 키워드	class, abstract, interface, extends, implements
객체 관련 키워드	new, instanceof, this, super, null
메소드 관련 키워드	void, return
제어문	if, else, switch, case, default, for, do, while, break, continue
논리 값	true, false
예외 처리	try, catch, finally, throw, throws
기타	transient, volatile, package, import!, synchronized, native, final, static, strictfp
사용되지 않는 키워드	goto, const

5. 변수와 상수 (Variables & Constants)

1. 프로그램은 각 식별자에 데이터를 저장하게 되는데 입력된 값을 다시 다른 값으로 변경할 수 있는 것을 변수, 안 되는 것을 상수라고 한다

변수 Variable

자료형 변수명 = 값 ; 의 형태로 정의되며
변수명 = 다른값 ; 의 형태로 값을 변경할 수 있다

```
String firstName = "Michael";

firstName = "Hong";
```

상수 Constant

변수명 앞에 final 을 붙여서 정의하고,
한번 입력된 값은 변경할 수 없다

```
final String firstName = "Michael";

firstName = "Hong";  // 에러
```

6. 기본 자료형 (Primitive Data Type)

자료형	내용
논리형 [boolean]	참, 거짓을 나타낸다 값 : true, false
문자형 [char]	문자입력 또는 Unicode '₩u0000' - ₩u 사용시 유니코드 'A' - 외따옴표 사용시 문자
정수형 [byte, short, int, long]	정수값 입력 * 기본적으로 int로 선언된다
실수형 [float, double]	소수점 자리를 입력할 수 있다 * 소수점 사칙연산시 BigDecimal 이용 권장 java.math.BigDecimal
문자형2 [String]	기본형은 아니지만 Java 에서 가장 많이 사용되는 데이터타입

7. 문자형 Data 타입의 적용

1. 하나의 문자를 기억하는 자료형으로 char형이 있다. char는 2byte의 메모리를 사용한다.

2. 유니코드는 2byte를 사용하여 한 문자를 처리하는 방식으로 거의 모든 나라의 언어를 처리할 수 있다. 자바는 문자를 처리하는 코드로 유니코드(Unicode)를 사용한다.

'글' char는 외 따옴표를 사용하여 값을 입력한다

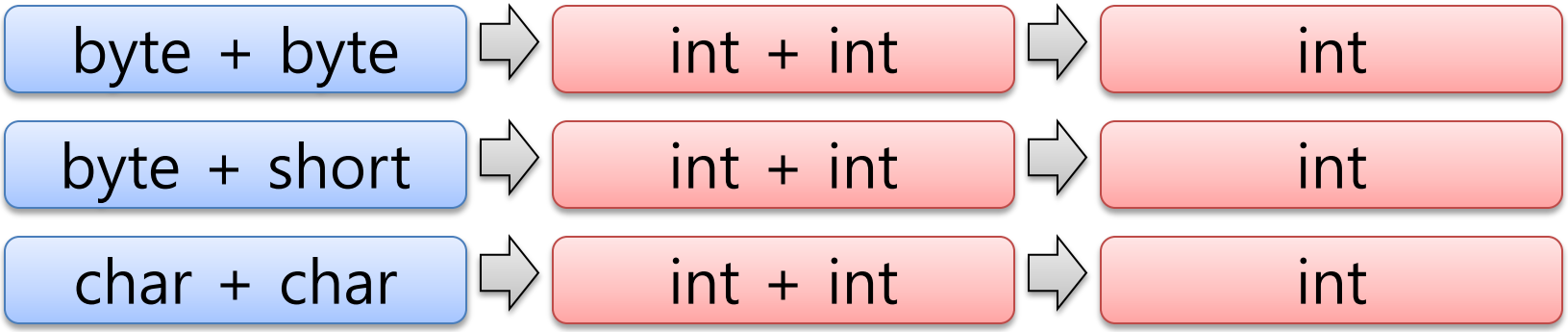
3. String 은 쌍 따옴표를 사용한다. String 은 내부적으로 char의 배열을 가진 형태의 객체이다

```
char word = '글';

String words = "한글";
// String words의 내부형태 = {
      char word[] = ['한', '글'];
}
```


8. 정수형 Data Type의 적용

정수형	메모리 양	정수 표현 범위
byte	1 byte	-128 ~ 127 (-2의 7승 ~ 2의 7승-1)
short	2 byte	-32768 ~ 32767 (-2의 15승 ~ 2의 15승-1)
int	4 byte	-2147483648 ~ 2147483647 (-2의 31승 ~ 2의 31승-1)
long	8 byte	-9223372036854775808L ~9223372036854775807L (-2의 63승 ~ 2의 63승-1)



9. 실수형 Data Type의 적용

실수형	메모리 양	실수 표현 범위
float	4 byte	-1.4E-45 ~ 3.4028235E+38
double	8 byte	-4.9E-324 ~1.7976931348623157E+308

1. 실수형 변수는 실수형 상수를 기억하기 위한 것으로 소수를 기억할 수 있다. 실수형 상수로는 float형과 double형이 있는데 **일반적인 소수는 모두 double형으로 간주**된다.

```
float f  = 2.56f;

double d = 2.56;

float f2 = 2.56;  // 에러 - 기본적으로 double 형으로 인식
```


10. 소수점 연산 오류 원인

1. 컴퓨터는 HW는 기본적으로 소수점인 0.1을 표현할 수 없다
   0 과 1의 정수만 표현가능한데, 소수점 표현을 위해 지수부와 가수부를 구분해서 표시하게 된다

> 지수가수 표기법
a = 0.1515 * 10E1

b = 0.55 * 10E-5

> 풀어쓰기
a = 1.515
b = 0.0000055

> 지수가수 표기법으로 변경 (가수에 할당된 자리수가 7자리일 경우)
a + b = 0.1515005~~5~~ * 10E1 < 1.5150055
> 여기서 0.0000005의 오차가 발생한다
> 가수부가 3자리로 한정되면 아래와 같이 뒷자리가 절삭된다
a + b = 0.152 * 10E1

11. 브랜드별 CPU 의 소수점 연산 오류

일명 거상 버그 (2006년)

아크코사인에 1.0 이상의 값 입력시

AMD	INTEL
1.000001	1.0000

12. 부동 소수점 연산의 한계

1. 각 언어별 국제 코딩표준에서는 정확한 수치연산의 경우 부동소수점 연산을 하지 못하도록 규정하고 있다

정확한 계산에서 소수점연산을 사용하는건 JSF나 MISRA-C의 규칙에 위배된다. JSF AV C++ 을 비롯한 많은 **코딩 표준**에서는 부동 소수점 변수를 직접적으로 비교하지 못하도록 규정하고 있다

```
public void compareDouble(double x, double y){
  if(x == y){
    System.out.println("X and Y is same ");
  }else{
    System.out.println("X and Y is not same");
  }
}
```

잘못된 코딩이 대표적인 예

```
if(Double.compare(x,y) == 0){
  // 로직 ...
}
```


13. 자료형변환 (Type Casting)

1. Java 에서는 계산 형태에 따라 아래처럼 자동으로 형을 변환해준다

변수1	변수2	결과 값
byte	byte	int
byte	short	int
int	byte	int
short	int	int
int	int	int
long	int	long

변수1	변수2	결과 값
long	long	long
byte	float	float
int	float	float
long	float	float
float	double	double
double	double	double

2. 그 이외의 경우는 강제로 형을 변환해서 사용해야 한다
   경우에 따라 값의 정밀도를 잃을 수 있다
   예) float -> int 변환시 소수점 이하 버림

```
int a = 10;
short b = (short) a;
```

강제로 변환하고자 하는
타입 명시 (타입)

14. 자료형변환 (문자<->숫자)

1. 숫자를 문자로 변경

```
int num = 2016;
String str = String.valueOf(num);

String str2 = num + "";
```

2. 문자를 숫자로 변경

```
String str = "2016";
byte b = Byte.parseByte(str);
short s = Short.parseShort(str);
int i = Integer.parseInt(str);
long l = Long.parseLong(str);
float f = Float.parseFloat(str);
double = Double.parseDouble(str);
```


15. 연산자 (Operators) 우선순위 1

Operator	Description	Level	Associativity
[] . () ++ --	access array element access object member invoke a method post-increment post-decrement	1	left to right
++ -- + - ! ~	pre-increment pre-decrement unary plus unary minus logical NOT bitwise NOT	2	right to left
() new	cast object creation	3	right to left
* / %	multiplicative	4	left to right
+ - +	additive string concatenation	5	left to right

15. 연산자 (Operators) 우선순위 2

Operator	Description	Level	Associativity
<< >> >>>	shift	6	left to right
< <= > >= instanceof	relational type comparison	7	left to right
== !=	equality	8	left to right
&	bitwise AND	9	left to right
^	bitwise XOR	10	left to right
|	bitwise OR	11	left to right
&&	conditional AND	12	left to right
||	conditional OR	13	left to right
?:	conditional	14	right to left
= += -= *= /= %= &= ^= |= <<= >>= >>>=	assignment	15	right to left

16. 산술 연산자

1. 일반적인 더하기, 빼기, 곱하기, 나누기, 그리고 나머지 연산자가 있다

```
int a = 2 + 3;                  //  a=5
int b = 2 - 3;                  //  b= -1
int c = 2 * 3;                  //  c= 6
int d = 2 / 3;                  //  d=0   소수점이하 반올림(자동형변환)
int e = 5 % 3;                  //  e=2    5를 3으로 나눈 나머지
int f = 10 % 3;                 //  f=1    10을 3으로 나눈 나머지
double dou = 5.0 % 4.2          //  dou=0.8
```


17. ! (not) 연산자

1. 논리적 NOT의 의미를 가지는 단항 연산자이다. 참(true)이면 거짓(false)으로, 거짓(false)이면 참(true)으로 바꾸는 역할을 한다.

```
boolean b1 = !true;      // b1에 false가 대입된다.
boolean b2 = !false;     // b2에 true가 대입된다.
```

아래의 !!와 같이 연속된 형태로 사용될 수 있다

```
boolean b= true;
System.out.println(!b);
System.out.println(!!b);
```

18. ~ (bit not) 연산자

1. 비트 논리적 NOT의 의미를 가지는 단항 연산자이다.
값을 2진 bit로 변경하고 각 bit의 값을 1은 0으로, 0은 1로 바꾸어 준다.

```
int a =7;
int b = ~a;
```

위 수식을 비트단위로 변경하면 아래와 같이된다

a	00000000 00000000 00000000 00000111 : 7
~a	11111111 11111111 11111111 11111000 : -8

```
// 연산 시 형 변환
byte a=7;
byte b=~a; //byte a 의 연산식인 ~a의 결과는 int로 담을 수 없다

byte b=(byte)(~a); // 강제로 형 변환이 필요하다
```


19. 쉬프트(Shift) 연산자 1

- 변수의 값을 2진 비트로 변경하고, 입력한 값만큼 비트를 이동시킨다

1. << 비트를 왼쪽으로 이동한다. 가장 오른쪽 비트는 0으로 채운다.

a ->	00000000 00000000 00000000 00000111 : 7
a<<1 ->	00000000 00000000 00000000 00001110 : 14

a ->	00000000 00000000 00000000 00000111 : 7
a<<2 ->	00000000 00000000 00000000 00011100 : 28

** 왼쪽으로 한번 쉬프트 할 때마다 값이 약 2배 증가한다*

2. >> 비트를 오른쪽으로 이동한다. 양수이면 가장 왼쪽 비트는 0으로 채우고 음수이면 1로 채운다

a ->	00000000 00000000 00000000 00000111 : 7
a>>1 ->	00000000 00000000 00000000 00000011 : 3

a ->	11111111 11111111 11111111 11001110 : -50
a>>1 ->	11111111 11111111 11111111 11100111 : -25

** 오른쪽으로 한번 쉬프트 할 때마다 값이 약 반으로 감소한다*

19. 쉬프트(Shift) 연산자 2

- 변수의 값을 2진 비트로 변경하고, 입력한 값만큼 비트를 이동시킨다

3. >>> 비트를 오른쪽으로 이동시키는데, 양수와 음수 모두 가장 왼쪽 비트는 0으로 채운다

**a -> **	00000000 00000000 00000000 00000111 : 7
a>>>1 ->	00000000 00000000 00000000 00000011 : 3

**a -> **	11111111 11111111 11111111 11001110 : -50
a>>>1 ->	01111111 11111111 11111111 11100111 : 2147483623

** >>> 연산자는 나누기 또는 곱하기 효과 보다는 비트 이동이 필요할 때 사용한다. 부호비트(가장왼쪽)에 0이 채워지므로 항상 양의 정수가 만들어진다*

20. 관계 연산자

1. 두 항의 비교를 통해 true 와 false인 boolean값을 출력한다

```
int a=1;
int b=2;
System.out.println(a==b);
System.out.println(a>b);
System.out.println(!(a>b));
System.out.println(b==2);
System.out.println(!(a!=b));
System.out.println();

System.out.println(true==false);
System.out.println(true==(1>0));
System.out.println(!!!!true);
```

* 체크 포인트 (= 이 하나일 경우와 == 두 개일 경우)
a=b : 변수 a에 b의 값을 대입한다.
a==b : a 와 b의 값을 비교한다. 같으면 true, 다르면 false이다.

21. 논리 연산자

1. &&(AND), | |(OR)
   두 개가 true 이거나, 둘 중 하나만 true 일 경우를 연산한다

A	B	A && B
true	true	true
true	false	false
false	true	false
false	false	false

[&& 연산 결과]

A	B	A || B
true	true	true
true	false	true
false	true	true
false	false	false

[\|\| 연산 결과]

*** Short-Circuit**

if (A && B) { ... }
A가 false이면, 이 식은 B와 관계없이 false가 되므로 B는 연산하지 않는다
if (A | | B) { ... }
A가 true이면 B와 관계없이 결과 값이 true가 되므로 B는 연산하지 않는다.

22. 비트 논리 연산자

1. & : 값을 2진 비트로 변환하고 비트끼리 AND 연산을 한다

a	1110
b	1100
a&b	1100

2. | : 값을 2진 비트로 변환하고 비트끼리 OR 연산을 한다

a	0011
b	1010
a|b	1011

3. ^ : 값을 2진 비트로 변환하고 비트끼리 Exclusive OR 연산을 한다

a	1110
b	1100
a^b	0010

*** 배타적 OR (Exclusive OR) 이란?**
비트끼리 비교해서 서로의 값이 다르면 true(1)가 리턴된다

23. 3항 연산자

1. 조건식 ? 값 1 : 값2 ;
   - 조건식이 참이면 값1을 리턴하고, 거짓이면 값2를 리턴한다

```
int a = 30;
int b = 15;
System.out.println(a > b ? true : false);    // true
```

2. 3항 연산자의 실 사용 예

```
boolean pass;
final int 국어_커트라인 = 80;
int 국어 = 85;
pass = ( 국어 > 국어_커트라인) ? true : false ;
```

```
// 위의 예를 일반 if 문으로 표현할 경우 구문이 길어진다
if ( 국어 > 국어_커트라인 ) {
    pass = true;
} else {
    pass = false;
}
```


24. 증감 연산자

1. ++, --

값을 1씩 더하고 빼는 연산자이다. 위치에 따라 결과가 달라진다

전위 연산자	++a, --a, ++b, --b

int b = ++a; 먼저 a에 1을 더하고 결과값을 b에 대입한다

후위 연산자	a++, a--, b++, b--

int b = a++; a를 먼저 b에 대입하고 a에 1을 더한다

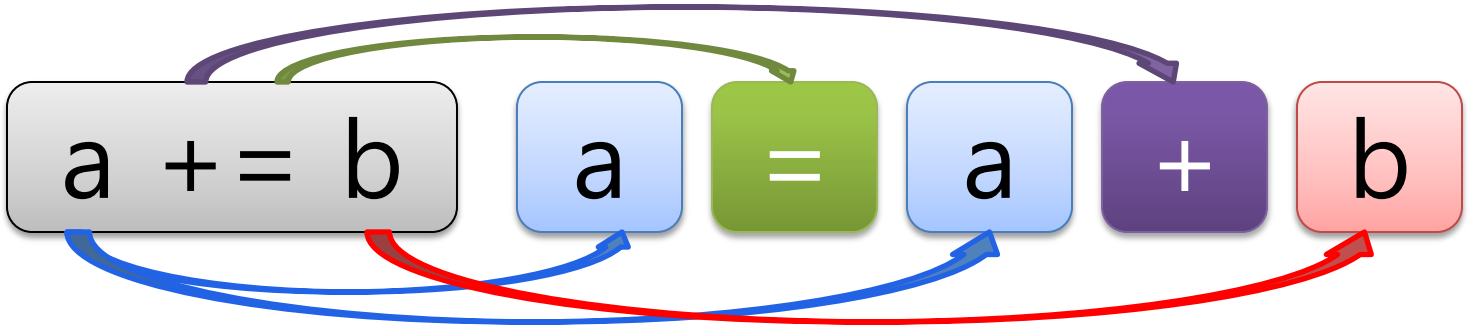
```
int a = 5, b=5;
int c = ++a;
int d = b++;
System.out.println("a = " + a);  // 6
System.out.println("b = " + b);  // 6
System.out.println("c = " + c);  // 6
System.out.println("d = " + d);  // 5
```


26. 대입 연산자

1. =
   순수하게 오른쪽의 값을 왼쪽에 대입한다

```
int a = 5;  // a 에 값 5를 대입한다
```

2. +=, -=, *=, /=, &=, ^=, |=
   왼쪽의 값과 오른쪽의 값을 연산한 후 왼쪽의 값에 대입한다




```
int a=6;
a += 3;    // 결과값  9  : a = a + 3 와 같다
a *= 3     // 결과값 18  : a = a * 3 와 같다
a -= 3     // 결과값  3  : a = a - 3 와 같다
a /= 3     // 결과값  2  : a = a / 3 와 같다
```


27. 미니 테스트

1. 마트에서 물건을 사고 거스름돈을 계산하는 프로그램을 작성해 보세요.
조건 : 입력값은 두 개(1. 지불금액, 2. 구입금액)
잔돈의 개수를 출력해야 합니다

입력조건)
지불금액 : 10,000원
구입금액 : 3,750원

총 거스름돈 : ?원
오천원 : ? 개
천원 : ?개
오백원 ?개
백원 : ?개
오십원 : ?개
십원 : ?개

