Chap 04. Expressions and Flow Control

학습 목표

1. 변수의 scope
2. Logical Operators
3. String Concatenation
4. Catsting 과 Promotion
5. 조건 분기문 (if, switch~case)
6. 순환문 (for, while, do~while)

❑ 변수의 분류

- 변수의 내용에 따라
  - Primitive type
  - Reference type

- 선언위치에 따라
  - 메소드 내부 선언, 메소드 파라미터 : Local Variable
  - 클래스 안 & 메소드 밖 선언 : Member Variable, Class Variable

❑ 변수의 생명 , 초기화 문제 , 참조 범위

- Local 변수
  - 메소드 시작시 생성, 메소드 종료시 해제 (stack)
  - 자동 초기화 안 된다. (반드시 사용전 명시적 초기화 필요)
  - 해당 메소드 내 참조
- 멤버 변수
  - 객체 생성시 생성, 객체 메모리 해제시 같이 해제 (heap)
  - 자동 초기화
  - 한 객체 내 참조
- 클래스 변수 (static)
  - 클래스 로드시 생성, Application 종료시 해제 (static 영역)
  - 자동 초기화
  - 전 클래스 객체 공통 참조

Local Variable의 초기화 문제

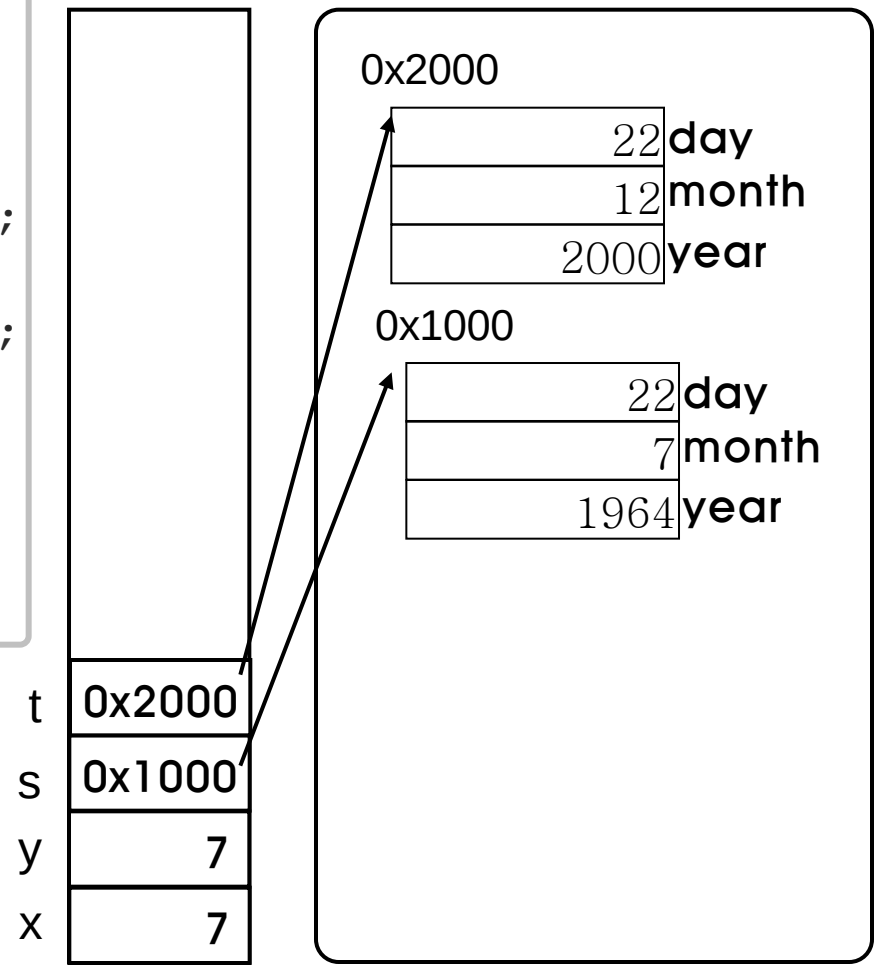

```
package chap04;

public class Initializing {

    public void doComputation() {

        int x = 10;
        int y;
        int z = 0;

        if ( x > 50 ) {
            y = 9;
        }

        z = y + x;
    }
}
```

Assigning References


```
int x = 7;

int y = x;

MyDate s = new MyDate( 22, 7, 1964 );

MyDate t = new MyDate( 22,12, 2000 );

s.setDay( 30 ); → 자신의 객체를 어떻게
                  찾을까?
```

Code 틀...

```
pubic void setDay( int day) {
    this.day = day;
}
```



Method Area

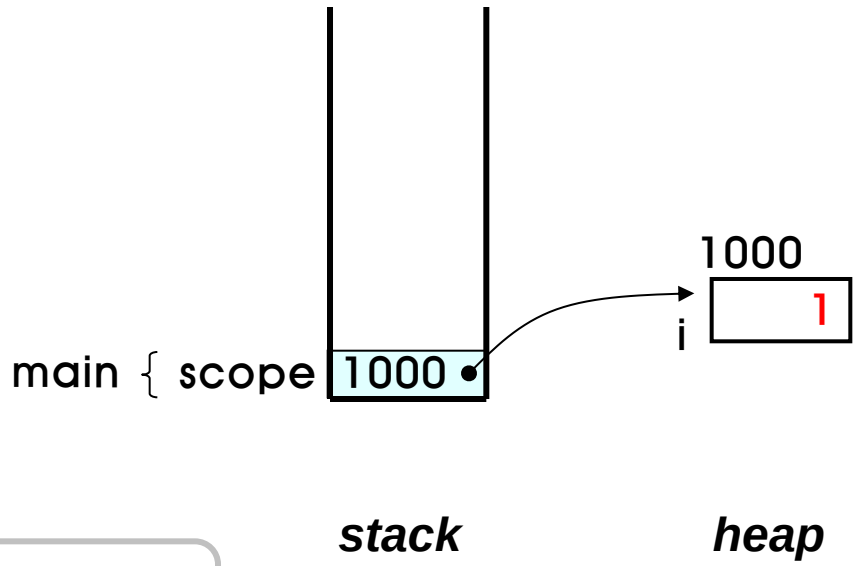
Stack

Heap

Scope Example


```
public class ScopeExample {
    private int i = 1;

    public void firstMethod() {
        int i = 4, j = 5;

        this.i = i + j;
        secondMethod( 7 );
    }

    public void secondMethod( int i ) {
        int j = 8;

        this.i = i + j;
    }
}
```

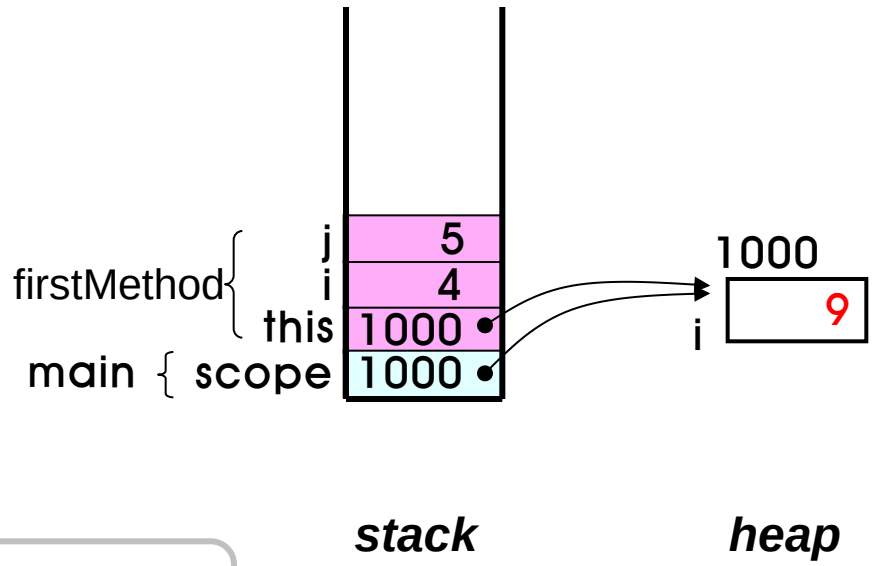
```
public class TestScoping {
    public static void main( String [] args ) {
        ScopeExample scope = new ScopeExample();
        scope.firstMethod();
    }
}
```



Scope Example

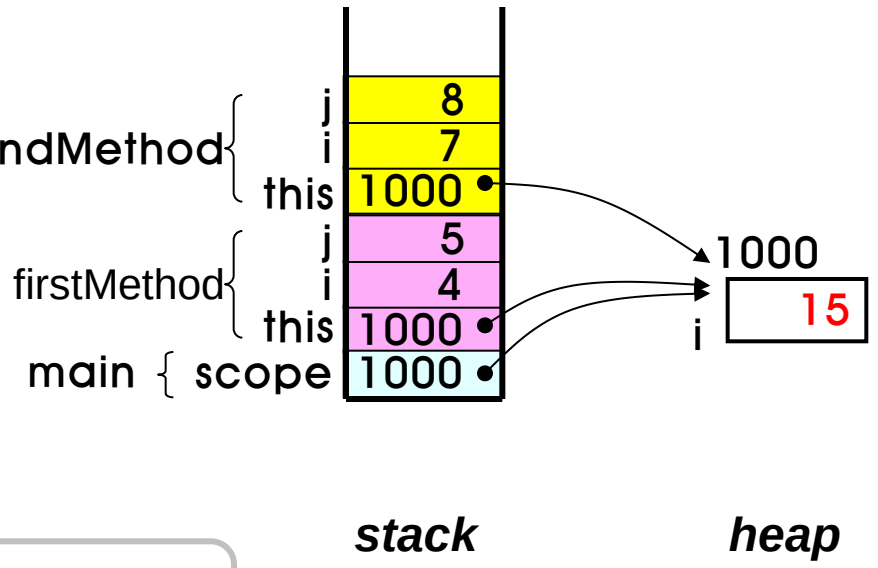

```
public class ScopeExample {
    private int i = 1;

    public void firstMethod() {
        int i = 4, j = 5;

        this.i = i + j;
        secondMethod( 7 );
    }

    public void secondMethod( int i ) {
        int j = 8;

        this.i = i + j;
    }
}
```

```
public class TestScoping {
    public static void main( String [] args ) {
        ScopeExample scope = new ScopeExample();
        scope.firstMethod();
    }
}
```



Scope Example


```
public class ScopeExample {
    private int i = 1;

    public void firstMethod() {
        int i = 4, j = 5;

        this.i = i + j;
        secondMethod( 7 );
    }

    public void secondMethod( int i ) {
        int j = 8;

        this.i = i + j;
    }
}
```




```
public class TestScoping {
    public static void main( String [] args ) {
        ScopeExample scope = new ScopeExample();
        scope.firstMethod();
    }
}
```

Java Operators

Operators	Associative	Precedence
++ -- + - ~ ! (*data_type*)	R to L	High ↕ Low
* / %	L to R	
+ -	L to R	
<< >> >>>	L to R	
< > <= >= instanceof	L to R	
== !=	L to R	
&	L to R	
^	L to R	
|	L to R	
&&	L to R	
||	L to R	
? :	R to L	
= *= /= %= += -= <<= >>= >>>= &= ^= |=	R to L	

Java Operators

- ✓ 사칙 연산 : + , − , * , /
- ✓ 나머지 연산 : %
- ✓ 단항 증감 연산자 : ++ , −−
- ✓ 같은지 비교 : ==
- ✓ 다른지 비교 : !=
- ✓ 크기 비교 : < , > , <= , >=
- ✓ 객체 비교 : instanceof
- ✓ 논리 연산 : & , | , && , || , !
- ✓ 타입 변환 : (*data_type*)

주의

```
int num = 10;
int result = 0;
1) result = num++;
   → result = num;
     num = num + 1;

2) result = ++num;
   → num = num + 1;
     result = num;
```

❑ Logical Operators

- ! : NOT
- & : AND
- | : OR

❑ Short-circuit Operators

- && : AND
- || : OR

A & B		Result
True	True	True
True	False	False
False	True	False
False	False	False

A && B		Result
True	True	True
True	False	False
False		False
False		False

A | B		Result
True	True	True
True	False	True
False	True	True
False	False	False

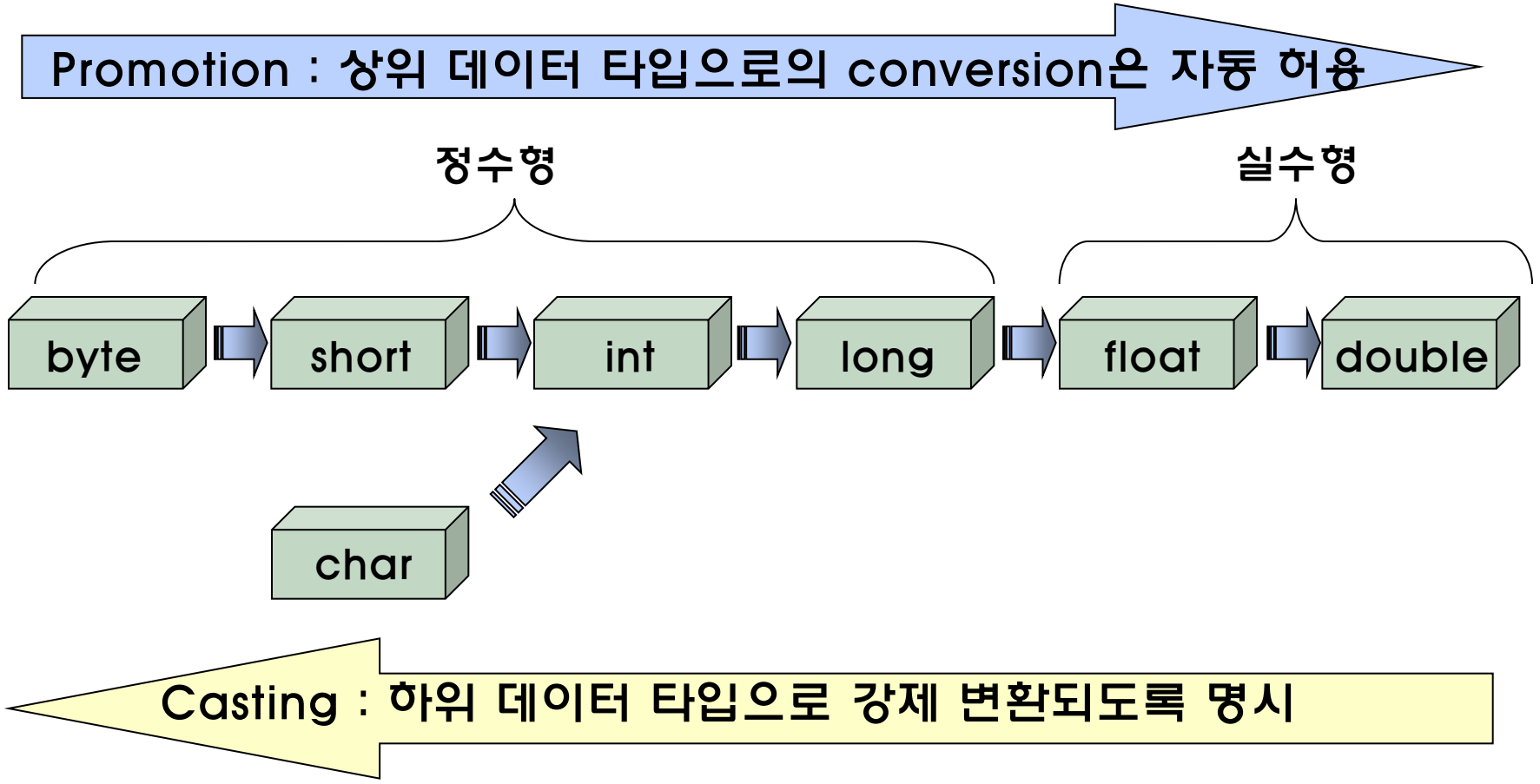
A || B		Result
True		True
True		True
False	True	True
False	False	False

! A	Result
True	False
False	True

Logical Operators Example


```
  int divisor = 0;
  int dividend = 100;

1)
  if ( divisor != 0 & dividend/divisor > 10 ) {
      System.out.println( "몫이 10보다 크다");
  }

2)
  if ( divisor != 0 && dividend/divisor > 10 ) {
      System.out.println( "몫이 10보다 크다");
  }
```

대원칙 >> 정보를 잃지 않는 방향으로 데이터 타입 간의 변환을 허용한다.



Promotion & Casting 예


```
long longVal = 6;   // OK (Promotion)
int intVal = 99L;   // compile error

double doubleVal = 12.414F; // OK (promotion)
float floatVal = 12.141;   // compile error
```

```
long longVal = 99L;
int intVal1 = longVal;     // compile error
int intVal2 = (int)longVal; //Casting 했으므로 OK(정보도 잃지 않음)
```

[주의]

```
int intVal = 128;
byte byteVal = (byte)intVal; // byteVal에 저장될 값 보장 못함
```

```
long longVal = 99L;
int intVal1 = longVal;        //error
int intVal2 = (int)longVal;   //OK

[주의]
   int intVal = 128;
   byte byteVal = (byte)intVal;
```

- 큰 데이터 타입을 작은 데이터 타입에 넣을 수 없다. → Compile Error
- Casting 연산자를 사용하여, 임시적으로 데이터 타입을 변환할 수 있다.
- Casting 연산자를 사용할 때에는, 적절한 범위의 값이 할당 되야 하는 것을 주의한다.
- Primitive data type 뿐만 아니라, Reference data type도 Type Casting이 가능하다.

Branching Statements – if

- expr 결과 값이 boolean인 문장들이나 식들이 올 수 있다.
- 중첩 if 문 사용시에는 대응 되는 else 문장을 주의 하여 작성한다.

```
if ( expr1 ) {
    statements;
}
```

```
if ( expr1 ) {
    statements1;
} else {
    statements2;
}
```

```
if ( expr1 ) {
    statements1;
} else if ( expr2 ) {
    statements2;
} else if ( expr3 ) {
    statements3,
else {
    statements4;
}
```

Branching Statements – if Example

```
public class IfTest {
    public static void main( String[] args ) {
        int i = 200;
        if ( i < 100 )
          if ( i > 10 )
            System.out.println( "10보다 크고 100보다 작다");
        else
            System.out.println( "i는 100이상이다.");
    }
}
```

Branching Statements – switch

- expr 은 int형으로 표현 될 수 있는 값이어야 한다. (byte, short, char, int)
- constant 는 int형 상수여야 한다.
- break 문이 없으면 fall-through 현상이 발생한다.

```
switch( expr ) {
    case constant1 :
            statements1;
            break;
    case constant2 :
            statements2;
            break;
    default :
            statements3;
            break;
}
```

Branching Statements – switch Example

```
public class SwitchTest {
    public static void main( String [] args ) {
        int aaa = 10;
        int bbb = 0;
        switch( aaa ) {
            case 5 :
                bbb = bbb + 1;
                break;
            case 10 :
                bbb = bbb + 2;
                break;
            case 15 :
                bbb = bbb + 4;
                break;
            default :
                bbb = bbb + 3;
                break;
        }
        System.out.println( bbb );
    }
}
```

Looping Statements

- Logic을 일정 조건 동안 반복적으로 수행하는 LOOP 구조에는 for, while, do~while 등 3가지가 있다.
- test_expr 의 결과는 boolean 타입 이어야 한다.

```
for ( init_expr ; test_expr ; alter_expr ) {
    Statements;
}
```

```
while ( test_expr ) {
    Statements;
}
```

```
do {
    Statements;
} while ( test_expr );
```

Looping Statements – Example

```
for ( int inx = 0 ; inx < 10 ; inx++ ) {
    System.out.println( inx );
}
```

```
int inx = 0;

while ( inx < 10 ) {
    System.out.println( inx );
    inx++;
}
```

```
int inx = 0;

do {
    System.out.println( inx );
    inx++;
} while ( inx < 10 );
```