

자바 기초 문법

자바 기초 문법

- ✚ 자바프로그램 작성, 실행
- ✚ 변수, 대입문
- ✚ 키워드, 식별자
- ✚ 기본 자료형
- ✚ 상수 변수, 리터럴
- ✚ 형 변환
- ✚ 연산자 및 우선순위
- ✚ if, for, while, break, continue
- ✚ 배열
- ✚ 향상된 for문
- ✚ 이차원배열, 다차원배열
- ✚ 메소드

자바프로그램 작성, 컴파일, 실행

소스 코드 작성 (Test.java)

```
public class Test {  
    public static void main(String[] args) {  
        System.out.println("안녕하세요");  
    }  
}
```

컴파일 (Test.class 생성)

```
javac Test.java
```

실행 (Test.class 실행)

```
java Test
```

변수(variable), 대입문(assignment)

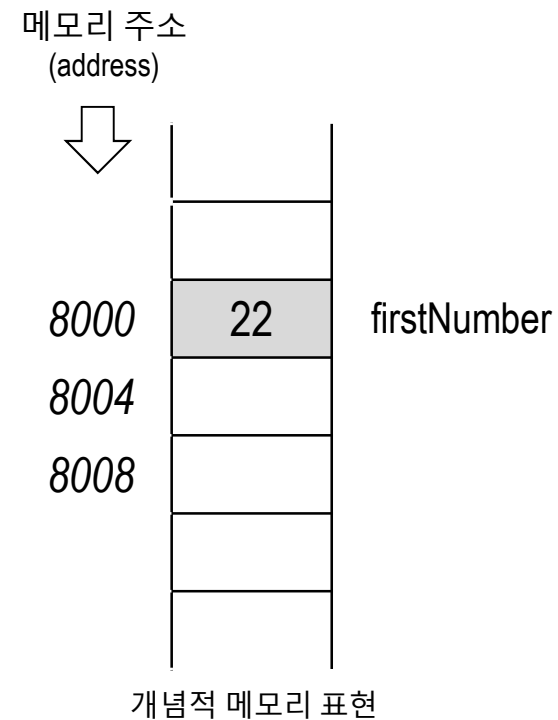
변수

- 이름이 부착된 메모리 내 저장 공간

자료형	변수명
↓	↓
int firstNumber; // 변수 선언/정의문	

```
firstNumber = 22; // 대입문
```

firstNumber와 연결된 메모리 저장소에 22를 대입(저장)



자바 프로그램 구조

클래스 Test 정의문

```
public class Test {
```

프로그램 실행은 main 메소드에서 시작됨

```
    public static void main(String[] args) {  
        int firstNumber=22; // 변수 선언 및 대입  
        int secondNumber=33; // 변수 선언 및 대입  
        int total=sum(firstNumber, secondNumber); // 메소드 호출  
        System.out.println("총합="+total); // 표준 출력  
    }
```

```
    private static int sum(int n, int m) {  
        int total=n+m;  
        return total;  
    }
```

```
}
```

메소드(method)
정의문

자바 프로그램 구조: 메소드 호출 및 리턴

```
public class Test {  
  
    public static void main(String[] args) {  
        int v1=22;  
        int v2=33;  
        int total = sum(v1, v2);  
        System.out.println("총합="+total);  
    }  
  
    private static int sum(int m, int n) {  
        int total = m + n;  
        return total;  
    }  
}
```

The diagram illustrates the flow of control between the `main` and `sum` methods. An arrow labeled "메소드 호출" (Method Call) points from the `sum(v1, v2)` call in the `main` method to the `sum` method definition. Another arrow labeled "리턴" (Return) points from the `return total;` statement in the `sum` method back to the `sum(v1, v2)` call in the `main` method. To the right of the `sum` method, the parameters are listed as `m = v1` and `n = v2`.

System.out.printf()

```
public class Test {  
    public static void main(String[] args) {  
        String    id="P-001";  
        int        age=35;  
        double     height=175.843;  
        char       gender='남';  
        boolean    foreignerYN=false;  
  
        System.out.printf("아이디=%s\n", id); // %s => 대응하는 값을 문자열로 교체  
        System.out.printf("나이=%d(세)\n", age); // %d => 대응하는 정수를 십진수로 교체  
        System.out.printf("신장=%f(cm)\n", height); // %f => 대응하는 실수를 십진수로 교체  
        System.out.printf("성별=%c\n", gender); // %c => 대응하는 문자로 교체  
        System.out.printf("외국인여부=%b\n", foreignerYN); // %b => 대응하는 불리언 값으로 교체  
  
        System.out.printf("아이디=%s, 나이=%d(세), 신장=%f(cm), 성별=%c, 외국인여부=%b\n", id, age, height, gender, foreignerYN);  
    }  
}
```

실행결과

```
아이디=P-001  
나이=35 (세)  
신장=175.843000 (cm)  
성별=남  
외국인여부=false  
아이디=P-001, 나이=35 (세), 신장=175.843000 (cm), 성별=남, 외국인여부=false
```

자바 키워드, 식별자

참조: https://docs.oracle.com/javase/tutorial/java/nutsandbolts/_keywords.html

자바 키워드(예약어, keywords, reserved words)

- abstract, continue, for, new, switch, assert, default, goto, package, synchronized, boolean, do, if, private, this, break, double, implements, protected, throw, byte, else, import, public, throws, case, enum, instanceof, return, transient, catch, extends, int, short, try, char, final, interface, static, void, class, finally, long, strictfp, volatile, const, float, native, super, while

식별자(identifier)

- 변수, 메소드, 클래스 등에 부여하는 이름
- 식별자 명명 규칙
 - ◆ 자바 키워드, true, false, null 사용 불가
 - ◆ 첫 문자로 숫자 사용 불가
 - ◆ 특수문자, 공백 사용 불가
 - \$, _는 사용 가능
 - ◆ 대소문자 구분 (case-sensitive)
 - ◆ 한글 사용 가능
 - ◆ 길이 제한 없음

```
public class Test {  
    public static void main(String[] args) {  
        int    score=10;  
        int    Score=20;  
        int    _score=30;  
        int    $score=40;  
    }  
}
```


자바의 기본 자료형

참조: <https://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html>

자바의 기본 자료형(primitive data type)

- 정수 기본 자료형 → int
- 실수 기본 자료형 → double

기본 자료형		바이트 크기	범위	필드 디폴트 값
정수	byte	1	$-2^7 \sim 2^7-1$	0
	short	2	$-2^{15} \sim 2^{15}-1$	0
	int	4	$-2^{31} \sim 2^{31}-1$	0
	long	8	$-2^{63} \sim 2^{63}-1$	0L
실수	float	4	32-bit IEEE-754	0.0f
	double	8	64-bit IEEE-754	0.0d
문자	char	2	16-bit Unicode	'\u0000'
불리언	boolean		true, false	false

리터럴

+ 정수 리터럴

+ 실수 리터럴

+ 문자 리터럴

● 예) '봄', 'A'

+ 불리언 리터럴

● true 혹은 false

+ 문자열(String) 리터럴

● 예) "서울", "A", "2", "3.4", "null", "true"

+ null 리터럴

● null

```
int    n = 12;  
long   m = 12L; // 12l  
int     v1 = 11; // 10진수 11  
int     v2 = 011; // 8진수 11 (10진수 9)  
int     v3 = 0x11; // 16진수 11 (10진수 17)  
int     v4 = 0b11; // 2진수 11 (10진수 3)
```

```
double  v = 12.0; // 12.0d, 12.0D  
float   w = 12.0F; // 12.0f  
double  x = 12E-3; // 12e-3, 0.012
```

```
char c1 = '한';  
char c2 = '\uD55C';  
char c3 = '\n';
```

```
boolean b = true; // true 혹은 false
```

```
String  s1 = "대한민국";  
String  s2 = null;
```

형 변환 (type conversion, type casting)

참조: <https://docs.oracle.com/javase/specs/jls/se7/html/jls-5.html>

✚ 확대 형 변환 (자동 수행됨)

● byte → short, char → int → long → float → double

✚ 축소 형 변환 (강제 수행 필요)

```
public class Test {  
    public static void main(String[] args) {  
        int    v = 123; // 4바이트  
        long   w = v; // 확대 형 변환 (4바이트 => 8바이트)  
        int    x = (int) w; // 축소 형 변환 (8바이트 => 4바이트)  
  
        double n = 3.14;  
        int    m = (int) n; // 축소 형 변환 (double => int)  
        System.out.println(m);  
    }  
}
```

연산자(operator)

유형	연산자	비고
증감	++ --	n++ ++n
산술	+ - * / %	
쉬프트	>> << >>>	
비교	> < >= <= == !=	
비트	& ^ ~	AND, OR, XOR, NOT
논리	&& ^ !	AND, OR, XOR, NOT
삼항	? :	(x>0)? x : -x
대입	= += -= *= /= &= ^= = <<= >>= >>>=	

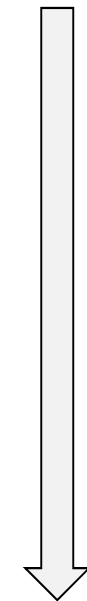
연산자 우선순위 (precedence)

참 조: <https://docs.oracle.com/javase/tutorial/java/nutsandbolts/operators.html>

참 조: <https://introcs.cs.princeton.edu/java/11precedence/>

유형	연산자	결합성(associativity)
Postfix 증감	n++ n--	
Prefix 증감, 양/음 부호, NOT	++n --n +n -n ~ !	R to L
곱셈, 나눗셈, 모듈로	* / %	L to R
덧셈, 뺄셈	+ -	L to R
쉬프트	<< >> >>>	L to R
비교	< > <= >= instanceof	
동등 비교	== !=	L to R
비트 단위 AND	&	L to R
비트 단위 XOR	^	L to R
비트 단위 OR		L to R
논리 AND	&&	L to R
논리 OR		L to R
삼항	? :	L to R
대입	= += -= *= /= %= &= ^= = <<= >>= >>>=	R to L

높음



낮음

조건문

if조건문

- 조건식의 참/거짓에 따라 문장의 1회 실행 여부가 결정되는 문장

```
if(조건식) {  
    조건식이 참일 때 1회 실행될 문장(들)  
}
```

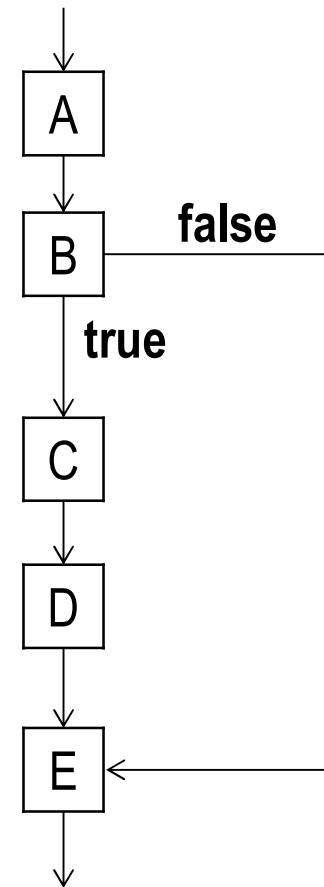
```
public class Test {  
    public static void main(String[] args) {  
        int age=18;  
        if(age<19) {  
            System.out.println("미성년자");  
        }  
    }  
}
```

```
if(조건식) {  
    조건식이 참일 때 1회 실행될 문장(들)  
} else {  
    조건식이 거짓일 때 1회 실행될 문장(들)  
}
```

```
public class Test {  
    public static void main(String[] args) {  
        int age=20;  
        if(age<19) {  
            System.out.println("미성년자");  
        } else {  
            System.out.println("성인");  
        }  
    }  
}
```

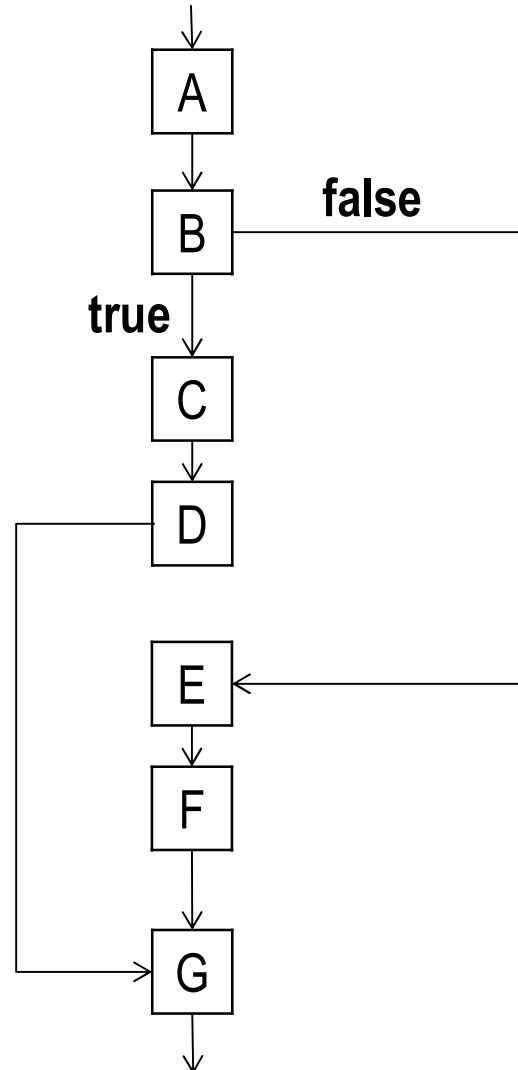
if 조건문 실행 흐름

```
A;  
if(B) {  
    C;  
    D;  
}  
E;
```



if-else 조건문 실행 흐름

```
A;  
if(B) {  
    C;  
    D;  
}  
else {  
    E;  
    F;  
}  
G;
```



조건식

+ 조건식

- 참, 거짓이 계산될 수 있는 식

+ 예

- `age < 20`
 - ◆ 변수 `age`에 저장된 값이 20보다 적으면 `true` 그렇지 않으면 `false`
- `age < 20 || age >= 65`
 - ◆ `age` 값이 20 미만이거나 `age` 값이 65 이상이면 `true`, 그렇지 않으면 `false`
- `age >= 20 && age < 65`
 - ◆ `age` 값이 20 이상이면서 65 미만이면 `true`, 그렇지 않으면 `false`
- `score == 100`
 - ◆ `score` 값이 100이면 `true`, 그렇지 않으면 `false`
- `score != 100`
 - ◆ `score` 값이 100이 아니면 `true`, 그렇지 않으면 `false`

while 반복문

while 반복문

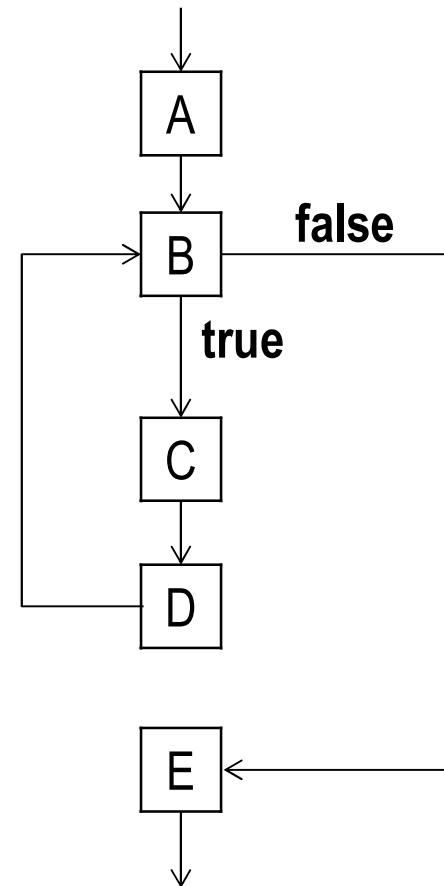
- 조건식이 참인 동안 반복적으로 실행되는 문장

```
while(조건식) {  
    조건식이 참일 동안 계속 반복적으로 실행될 문장(들)  
}
```

```
public class Test {  
    public static void main(String[] args) {  
        int i=1;  
        int sum=0;  
        while (i<=5){  
            sum=sum + i;  
            i+=1;  
        }  
        System.out.println(sum);  
    }  
}
```

while 반복문 실행 흐름

```
A;  
while(B) {  
    C;  
    D;  
}  
E;
```



for 반복문

for 반복문

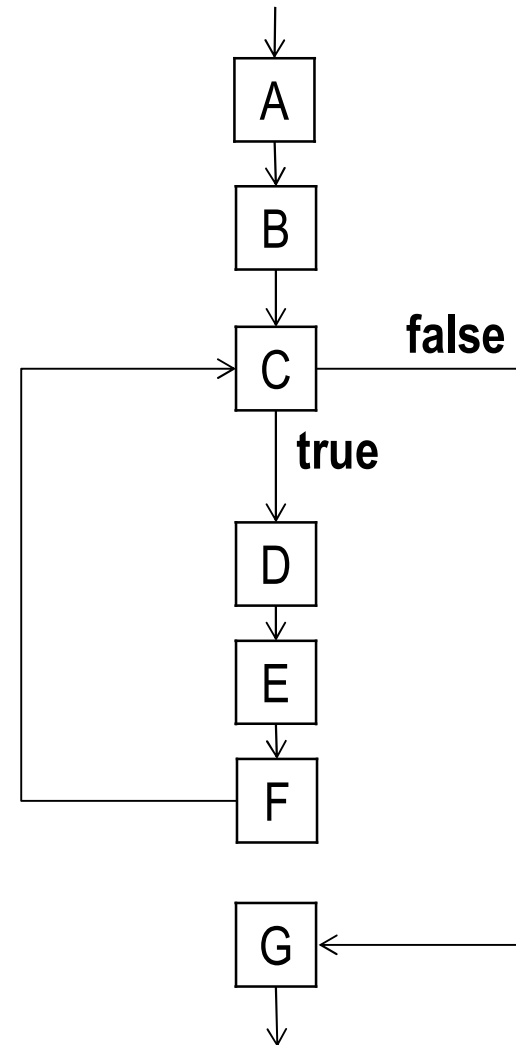
- 조건식이 참인 동안 반복적으로 실행되는 문장

```
for(최초 1회 실행될 문장 ; 조건식 ; 블록 실행 후 실행될 문장)
{
    조건식이 참일 동안 계속 반복적으로 실행될 문장(들)
}
```

```
public class Test {
    public static void main(String[] args) {
        int sum=0;
        for (int i = 1; i <= 5; i++) {
            sum=sum + i;
        }
        System.out.println(sum);
    }
}
```

for 반복문 실행 흐름

```
A;  
for(B;C;F)  
{  
    D;  
    E;  
}  
G;
```



반복 탈출문: break

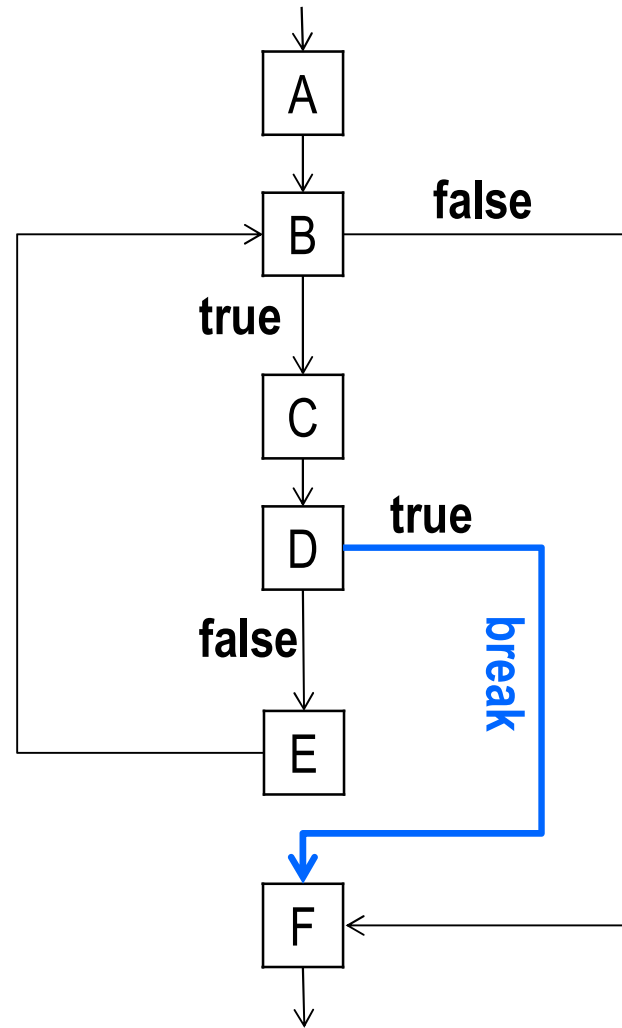
break

- break 문이 실행되면 이 break문을 포함하는 반복문을 탈출

```
public class Test {  
    public static void main(String[] args) {  
        int i=1;  
        int sum=0;  
        while(true){  
            if(i>5) break;  
            sum=sum + i;  
            i+=1;  
        }  
        System.out.println(sum);  
    }  
}
```

break 문 실행 흐름

```
A;  
while(B) {  
    C;  
    if(D) break;  
    E;  
}  
F;
```



반복 계속문: continue

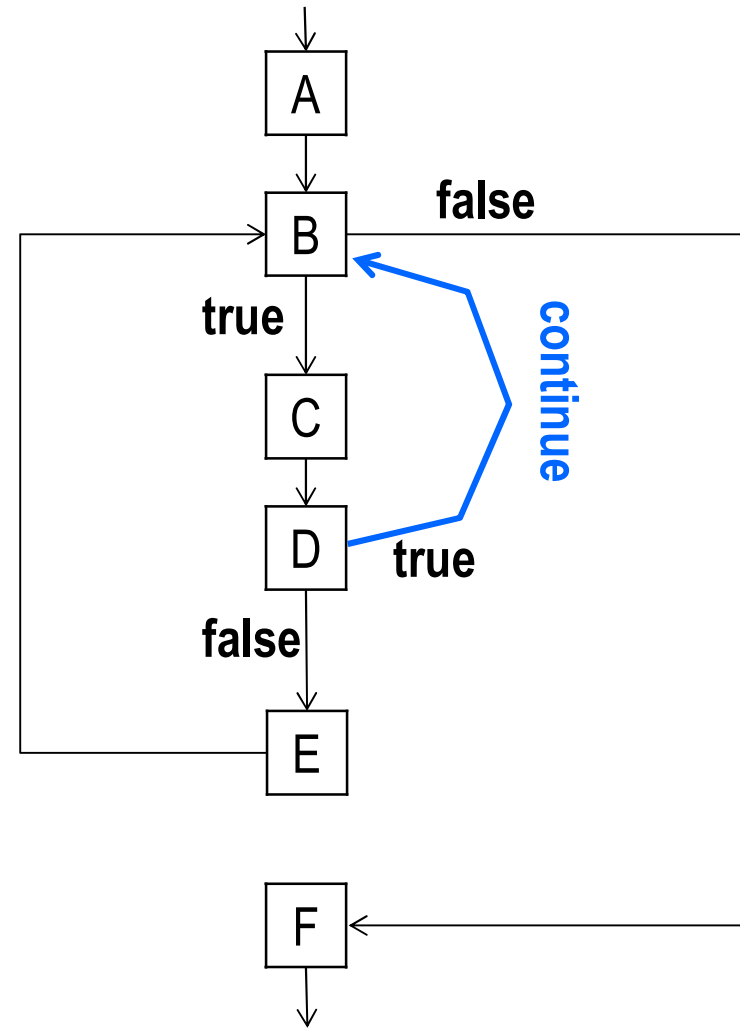
continue

- continue 문이 실행되면 이 continue문 이후 반복문을 무시하고 반복 절차를 다시 계속

```
public class Test {  
    public static void main(String[] args) {  
        int sum=0;  
        for (int i = 1; i <= 100; i++) {  
            if(i%2==1) continue;  
            sum=sum + i;  
        }  
        System.out.println(sum);  
    }  
}
```

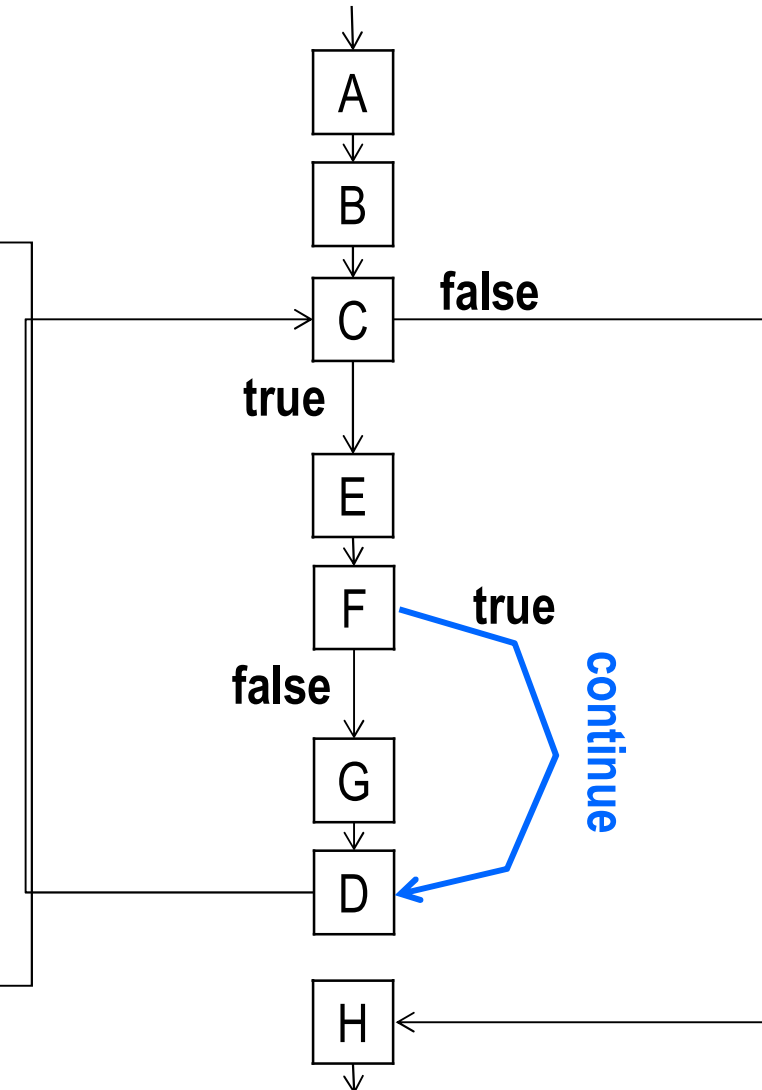

while 반복문의 continue 문 실행 흐름

```
A;  
while(B) {  
    C;  
    if(D) continue;  
    E;  
}  
F;
```



for 반복문의 continue 문 실행 흐름

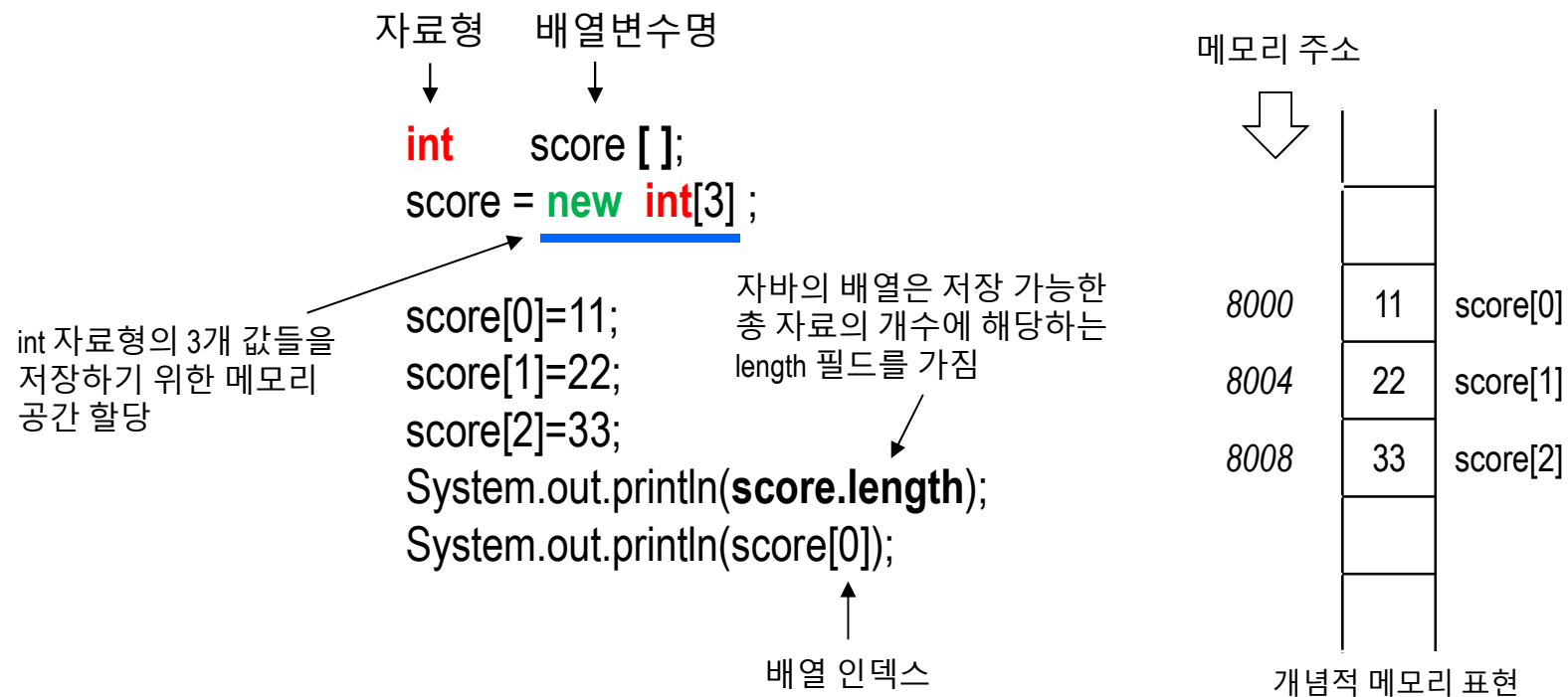
```
A;  
for(B;C;D) {  
    E;  
    if(F) continue;  
    G;  
}  
H;
```



배열(array)

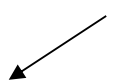
배열

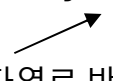
- 인덱스(index)로 접근 가능한 동일 자료형 값들을 연속된 메모리 공간에 저장할 수 있는 자료구조



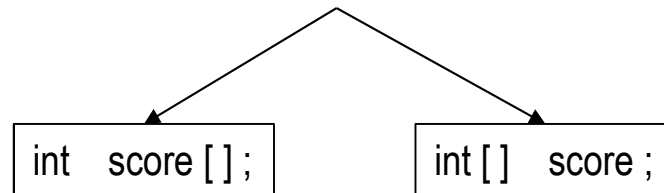
배열 생성, 초기화

```
public class Test {  
    public static void main(String[] args) {  
        int    score[ ];  
        score = new int[3];  
        score[0]=11;  
        score[1]=22;  
        score[2]=33;  
        for (int i = 0; i < score.length; i++) {  
            System.out.println(score[i]);  
        }  
    }  
}
```

```
public class Test {  
    public static void main(String[] args) {  
        int    score[ ] = new int[3];  
        score[0]=11;  
        score[1]=22;  
        score[2]=33;  
        for-each 문  
        for (int v : score) {  
            System.out.println(v);  
        }  
    }  
}
```

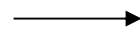
```
public class Test {  
    public static void main(String[] args) {  
        int    score[] = {11,22,33};  
        System.out.println(Arrays.toString(score));  
    }  
} 배열 내 자료들을 문자열로 반환하는 메소드
```

배열 변수 선언 시 아래 두 방법 모두 가능



for-each 문, enhanced for 문

배열 score 내 인덱스 0부터 인덱스
score.length-1까지 각 원소 값에 대해
그 값을 변수 v에 대입한 후 반복문
블록을 실행한다.



```
int    score[] = {11, 22, 33};  
  
for (int v : score) {  
    System.out.println(v);  
}
```

이차원배열 (two-dimensional array)

	0	1	2
0	11	22	33
1	44	55	66

```
public class Test {
    public static void main(String[] args) {
        int v[] [] = new int[2][3];
        v[0][0]=11;
        v[0][1]=22;
        v[0][2]=33;
        v[1][0]=44;
        v[1][1]=55;
        v[1][2]=66;
        System.out.println(v.length); // 행 크기
        System.out.println(v[0].length); // 0번째 행의 열 크기
        System.out.println(v[1].length); // 1번째 행의 열 크기
        for (int i = 0; i < v.length; i++) {
            for (int j = 0; j < v[i].length; j++) {
                System.out.print(v[i][j]+" ");
            }
            System.out.println();
        }
    }
}
```

int v[][] = { {11,22,33}, {44,55,66} };

이차원배열 (two-dimensional array)

	0	1	2	3
0	11	22	33	44
1	55	66		
2	77	88	99	

```
public class Test {  
    public static void main(String[] args) {
```

```
        int v[ ][ ] = new int[3][ ];
```

```
        v[0]=new int[4];
```

```
        v[1]=new int[2];
```

```
        v[2]=new int[3];
```

```
        v[0][0]=11;
```

```
        v[0][1]=22;
```

```
        v[0][2]=33;
```

```
        v[0][3]=44;
```

```
        v[1][0]=55;
```

```
        v[1][1]=66;
```

```
        v[2][0]=77;
```

```
        v[2][1]=88;
```

```
        v[2][2]=99;
```

```
        System.out.println(v.length); // 행 크기
```

```
        System.out.println(v[0].length); // 0번째 행의 열 크기
```

```
        System.out.println(v[1].length); // 1번째 행의 열 크기
```

```
        System.out.println(v[2].length); // 2번째 행의 열 크기
```

```
        System.out.println(Arrays.toString(v[0]));
```

```
        System.out.println(Arrays.toString(v[1]));
```

```
        System.out.println(Arrays.toString(v[2]));
```

```
    }
```

```
}
```

```
int v[ ][ ] = { {11,22,33,44}, {55,66}, {77,88,99} };
```

다차원배열 (multi-dimensional array)

```
public class Test {  
    public static void main(String[] args) {  
        int v[][][] = new int[10][40][3];  
        System.out.println(v.length);  
        System.out.println(v[0].length);  
        System.out.println(v[0][0].length);  
        v[0][7][1]=95;  
        System.out.println(Arrays.toString(v[0][7]));  
    }  
}
```

- 10개 학급
- 학급별 최대 학생 40명
- 각 학생의 국영수 3개 과목 점수

메소드 (method)

메소드

- 클래스 내에 정의되는 함수

메소드 수식어(modifier) **public static**

- public** → Test가 아닌 다른 클래스로부터 호출 가능
- static** → Test 클래스의 객체에 무관하게 호출됨

반환 자료형 **void** (없음)

메소드 이름 **main**

괄호 내 파라미터 리스트 **String [] args**

public class Test {

- 메소드 블록 {...}
- 메소드 코드 부분

```
public static void main(String[] args)
{
    System.out.println("안녕하세요");
}
```

main 메소드 정의문

메소드 (method)

```
public class Test {
```

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

```
        int v1 = 22;
```

```
        int v2 = 33;
```

```
        int total = sum(v1,v2) ;
```

```
    }
```

sum 메소드
호출문

메소드 수식어(modifier) **private static**

- **private** → Test 클래스 내에서만 호출 가능
- **static** → Test 클래스의 객체에 무관하게 호출됨

반환 자료형 **int**

메소드 이름 **sum**

괄호 내 파라미터 리스트 **int n, int m**

```
        private static int sum(int n, int m)
```

```
        {
```

```
            int total = n + m;
```

```
            return total; // int 자료형 값 반환
```

```
        }
```

메소드 블록

```
    }
```

sum 메소드 정의문

메소드 (method)

```
public class Test {  
    public static void main(String[] args) {  
        int n[] = {1,2,3,4,5};  
        int total = sum(n);  
        System.out.println(total);  
    }  
}
```

정수 배열 **n**을 메소드
인자(argument)로 전달

정수 배열을 메소드
파라미터(parameter)로 전달
받음 **int [] n**

```
private static int sum(int[] n) {  
    int total = 0;  
    for (int i = 0; i < n.length; i++) {  
        total += n[i];  
    }  
    return total; // int 자료형 값 반환  
}
```

정수 배열을 파라미터로
전달 받아 배열 내 정수들의
총합을 정수로 반환하는
메소드 sum()

메소드 (method)

```
public class Test {  
    public static void main(String[] args) {  
        int n[] = {1,2,3};  
        int m[] = {4,5};  
        int v[] = concatArray(n,m);  
        for (int i = 0; i < v.length; i++) {  
            System.out.println(v[i]);  
        }  
    }  
}
```

정수 배열 **n, m**을 메소드
인자(argument)로 전달

두 정수 배열을 메소드
파라미터(parameter)로 전달
받음 **int [] n, int [] m**


두 정수 배열을 파라미터로
전달 받아 두 배열 내 각
정수들을 하나의 정수 배열
v에 저장한 후 배열 v를
반환하는 메소드

```
private static int[] concatArray(int[] n, int[] m) {  
    int v[] = new int[ n.length + m.length ];  
    int k=0;  
    for (int i = 0; i < n.length; i++) { v[k++] = n[i]; }  
    for (int i = 0; i < m.length; i++) { v[k++] = m[i]; }  
    return v; // 정수 배열 int [] 반환  
}
```

References

 <http://docs.oracle.com/javase/7/docs/api/>

 <https://docs.oracle.com/javase/tutorial/java/>

 김윤명. (2008). 뇌를 자극하는 Java 프로그래밍. 한빛미디어.

 남궁성. 자바의 정석. 도우출판.

 황기태, 김효수 (2015). 명품 Java Programming. 생능출판사.