

제04장

제어문

구디아카데미 ▷ 민경태 강사

```
each: function(e, t, n) {  
  var r, i = 0,  
      o = e.length,  
      a = M(e);  
  if (n) {  
    if (a) {  
      for (; o > i; i++)  
        if (r = t.apply(e[i], n), r ===  
    } else  
      for (i in e)  
        if (r = t.apply(e[i], n), r ===  
  } else if (a) {  
    for (; o > i; i++)  
      if (r = t.call(e[i], i, e[i]))  
    } else  
      for (i in e)  
        if (r = t.call(e[i], i, e[i]))  
    return e  
  },  
  trim: b && !b.call("\uffff\u00a0") ?  
    return null == e ? "" : b.call(  
  } : function(e) {  
    return null == e ? "" : (e + "  
  },  
  makeArray: function(e, t) {  
    var n = t || [];  
    return null != e && (M(Obj  
  },  
  isArray: function(e, t, n) {  
    var r;  
    if (t) {  
      if (n) return n.c  
      for (n = t.length;  
        if (n in t)  
    }  
  }
```

학습목표

1. 분기문에 대해서 알 수 있다.
2. 반복문에 대해서 알 수 있다.
3. 기타 제어문에 대해서 알 수 있다.

```
each: function(e, t, n) {  
  var r, i = 0,  
      o = e.length,  
      a = M(e);  
  if (n) {  
    if (a) {  
      for (; o > i; i++)  
        if (r = t.apply(e[i], n), r ===  
    } else  
      for (i in e)  
        if (r = t.apply(e[i], n), r ===  
  } else if (a) {  
    for (; o > i; i++)  
      if (r = t.call(e[i], i, e[i])  
  } else  
    for (i in e)  
      if (r = t.call(e[i], i, e[i])  
  return e  
},  
trim: b && !b.call("\uffff\u00a0") ?  
  return null == e ? "" : b.call(  
} : function(e) {  
  return null == e ? "" : (e +  
},  
makeArray: function(e, t) {  
  var n = t || [];  
  return null != e && (M(Obj  
},  
isArray: function(e, t, n) {  
  var r;  
  if (t) {  
    if (n) return m.c  
    for (r = t.length  
      if (n in t  
  }  
}
```

목차

1. 분기문

- 1) if문
- 2) else if문
- 3) else문
- 4) switch문과 break문

2. 반복문

- 1) while문
- 2) do while문
- 3) for문
- 4) 반복문의 중첩

```
each: function(e, t, n) {  
  var r, i = 0,  
      o = e.length,  
      a = M(e);  
  if (n) {  
    if (a) {  
      for (; o > i; i++)  
        if (r = t.apply(e[i], n), r ===  
    } else  
      for (i in e)  
        if (r = t.apply(e[i], n), r ===  
  } else if (a) {  
    for (; o > i; i++)  
      if (r = t.call(e[i], i, e[i]))  
    } else  
      for (i in e)  
        if (r = t.call(e[i], i, e[i]))  
    return e  
  },  
  trim: b && !b.call("\uffff\u00a0") ?  
    return null == e ? "" : b.call(  
  } : function(e) {  
    return null == e ? "" : (e + "  
  },  
  makeArray: function(e, t) {  
    var n = t || [];  
    return null != e && (M(Obj  
  },  
  isArray: function(e, t, n) {  
    var r;  
    if (t) {  
      if (n) return n.c  
      for (n = t.length;  
        if (n in t)  
    }  
  }
```

목차

3. 기타 제어문

- 1) break문
- 2) continue문

```

each: function(e, t, n) {
    var r, i = 0,
        o = e.length,
        a = M(e);
    if (n) {
        if (a) {
            for (; o > i; i++)
                if (r = t.apply(e[i], n), r === !1) break;
        } else
            for (i in e)
                if (r = t.apply(e[i], n), r === !1) break;
    } else if (a) {
        for (; o > i; i++)
            if (r = t.call(e[i], e[i]), r === !1) break;
    } else
        for (i in e)
            if (r = t.call(e[i], e[i]), r === !1) break;
    return e;
},
trim: b && !b.call("\uffff\u00a0") ? function(e) {
    return null == e ? "" : b.call(e);
} : function(e) {
    return null == e ? "" : (e + "").replace(C, "");
},
makeArray: function(e, t) {
    var n = t || [];
    return null != e && (M(Object(e)) ? x.merge(n, "string"
),
isArray: function(e, t, n) {
    var r;
    if (t) {
        if (n) return m.call(t, e, n);
        for (r = t.length, r = n ? 0 > n ? Math.max(0, r + n) : r; r--;)
            if (n in t && t[n] === e) return n;
    }
}

```

1. 분기문

- 어떤 조건을 만족하는 경우에만 실행하는 코드를 작성할 때 사용함
- 조건식은 반드시 boolean 자료형의 결과(true 또는 false)를 반환해야 함
- 조건을 만족할 때 실행할 코드는 중괄호{}로 묶어서 다른 코드와 구분함
- 조건을 만족할 때 실행할 코드가 1개인 경우 중괄호{}는 생략할 수 있음

if문 형식

■ 형식

```
if (조건식) {  
    실행문1  
    실행문2  
}
```

조건식의 결과가 true인 경우에 실행문1, 실행문2를 실행한다.

```
if (조건식) {  
    실행문  
}
```

실행문이 하나인 경우 중괄호{}는 생략 가능하다.

if문 예시

- 점수가 60점 이상이면 "승인"

```
int score = 100;  
if (score >= 60) {  
    System.out.println("승인");  
}
```

- 나이가 10살 이상 20살 미만이면 "10대"

```
int age = 15;  
if (age >= 10 && age < 20) {  
    System.out.println("10대");  
}
```


else if문

- if문 이후에 추가할 수 있는 선택 구문
- if문의 조건을 만족하지 못한 경우를 대상으로 새로운 조건을 지정할 때 사용함
- if문처럼 조건을 만족할 때 실행할 코드가 2개 이상이면 중괄호{}로 묶고 1개이면 생략할 수 있음
- else if문은 2개 이상 추가할 수 있음

else if문 형식

■ 형식

```
if (조건식1) {  
    실행문1  
    실행문2
```

조건식1의 결과가 true인 경우에 실행문1, 실행문2를 실행한다.

```
} else if (조건식2) {  
    실행문1  
    실행문2
```

조건식1의 결과가 false인 경우에 조건식2를 실행한다.

```
} else if (조건식3) {  
    실행문1  
    실행문2  
}
```

조건식2의 결과가 false인 경우에 조건식3을 실행한다.

else if문 예시

- 점수가 60점 이상이면 "승인", 50점 이상이면 "조건부승인"

```
int score = 55;
if (score >= 60) {
    System.out.println("승인");
} else if(score >= 50) {
    System.out.println("조건부승인");
}
```

else문

- if문의 마지막에 추가할 수 있는 선택 구문
- if문이나 else if문의 조건을 모두 만족하지 못한 경우에 처리할 코드를 담당함
- if문이나 else if문처럼 조건을 만족할 때 실행할 코드가 2개 이상이면 중괄호{}로 묶고 1개이면 생략할 수 있음
- if문이나 else if문과 달리 조건식을 작성할 수 없음

else문 형식

■ 형식

```
if (조건식1) {  
    실행문1  
    실행문2  
} else if (조건식2) {  
    실행문1  
    실행문2  
} else {  
    실행문1  
    실행문2  
}
```

조건식1의 결과가 true인 경우에 실행문1, 실행문2를 실행한다.

조건식1의 결과가 false인 경우에 조건식2를 실행한다.

조건식1, 조건식2의 결과가 모두 false인 경우에 실행문1, 실행문2를 실행한다.

else문 예시

- 점수가 60점 이상이면 "승인", 50점 이상이면 "조건부승인", 나머지는 "미승인"

```
int score = 55;
if (score >= 60) {
    System.out.println("승인");
} else if(score >= 50) {
    System.out.println("조건부승인");
} else {
    System.out.println("미승인");
}
```

switch문

- 어떤 코드부터 실행을 시작할지 결정할 때 사용하는 구문
- case문을 여러 개 배치한 뒤 원하는 case문부터 실행할 수 있음
- switch문에 작성한 표현식을 실행한 뒤 해당 표현식의 결과값에 따라 실행을 시작할 case문이 선택됨
- 표현식의 결과값은 byte, short, int, char, String 타입 중 하나만 가능함 (boolean, long, float, double 불가능)

switch문 형식

■ 형식

```
switch(표현식) {  
  case 값1:  
    실행문1;  
  case 값2:  
    실행문2;  
  case 값3:  
    실행문3;  
  default:  
    실행문4;  
}
```

표현식 == 값1이면 실행문1, 실행문2, 실행문3, 실행문4를 실행한다.

표현식 == 값2이면 실행문2, 실행문3, 실행문4를 실행한다.

표현식 == 값3이면 실행문3, 실행문4를 실행한다.

나머지 경우에는 실행문4를 실행한다.

break문

- switch문의 실행을 종료할 때 사용함
- 원하는 case문만 실행하려면 case문이 끝나기 전에 break문을 삽입하여 다음 case문을 실행하는 것을 방지할 수 있음

break문 형식

■ 형식

```
switch(표현식) {  
  case 값1:  
    실행문1; break;  
  case 값2:  
    실행문2; break;  
  case 값3:  
    실행문3; break;  
  default:  
    실행문4;  
}
```

표현식 == 값1이면 실행문1을 실행하고 switch문을 종료한다.

표현식 == 값2이면 실행문2을 실행하고 switch문을 종료한다.

표현식 == 값3이면 실행문3을 실행하고 switch문을 종료한다.

나머지 경우에는 실행문4를 실행하고 switch문을 종료한다.

switch문 예시

- 등급이 "GOLD"이면 300포인트, "SILVER"이면 200포인트, "BRONZE"이면 100포인트, 나머지 등급은 50포인트

```
String grade = "GOLD";  
int point = 0;  
switch(grade) {  
case "GOLD":  
    point = 300; break;  
case "SILVER":  
    point = 200; break;  
case "BRONZE":  
    point = 100; break;  
default:  
    point = 50;  
}  
System.out.println(point);
```

```

each: function(e, t, n) {
    var r, i = 0,
        o = e.length,
        a = M(e);
    if (n) {
        if (a) {
            for (; o > i; i++)
                if (r = t.apply(e[i], n), r === !1) break;
        } else
            for (i in e)
                if (r = t.apply(e[i], n), r === !1) break;
    } else if (a) {
        for (; o > i; i++)
            if (r = t.call(e[i], e[i]), r === !1) break;
    } else
        for (i in e)
            if (r = t.call(e[i], e[i]), r === !1) break;
    return e;
},
trim: b && !b.call("\uffff\u00a0") ? function(e) {
    return null == e ? "" : b.call(e);
} : function(e) {
    return null == e ? "" : (e + "").replace(C, "");
},
makeArray: function(e, t) {
    var n = t || [];
    return null != e && (M(Object(e)) ? x.merge(n, "string"
),
isArray: function(e, t, n) {
    var r;
    if (t) {
        if (n) return m.call(t, e, n);
        for (r = t.length, r = r ? 0 > r ? Math.max(0, r + n) : r; r--;)
            if (n in t && t[n] === e) return n;
    }
}

```

2. 반복문

while문

- 어떤 조건을 만족(true)하는 경우 반복해서 여러 번 실행하는 코드를 작성함
- 반복 실행하는 코드는 중괄호{}로 묶어서 다른 코드와 구분함
- 반복 실행하는 코드가 하나인 경우 중괄호{}는 생략 가능함

while문 형식

■ 형식

```
while (조건식) {  
    실행문1  
    실행문2  
}
```

조건식의 결과가 true인 경우에 실행문1, 실행문2를 처리한 뒤 다시 조건식을 실행한다.

```
while (조건식) {  
    실행문  
}
```

반복할 실행문이 하나인 경우 중괄호{}는 생략 가능하다.

while문 코드 분석1

- 0 1 2 출력하기

```
int a = 0;
```

a는 0을 가진 상태

```
while (a < 3) {  
    System.out.println(a);  
    a++;  
}
```

0 < 3 이므로 중괄호{} 내부를 실행한다.

while문 코드 분석2

- 0 1 2 출력하기

```
int a = 0;
```

```
while (a < 3) {  
    System.out.println(a);  
    a++;  
}
```

0을 출력하고 a를 1 증가시킨다.(a는 1을 가진 상태)

while문 코드 분석3

- 0 1 2 출력하기

```
int a = 0;
```

```
while (a < 3) {  
    System.out.println(a);  
    a++;  
}
```

1 < 3 이므로 중괄호{} 내부를 실행한다.

while문 코드 분석4

- 0 1 2 출력하기

```
int a = 0;
```

```
while (a < 3) {
```

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

```
    a++;
```

```
}
```

1을 출력하고 a를 1 증가시킨다.(a는 2를 가진 상태)

while문 코드 분석5

- 0 1 2 출력하기

```
int a = 0;
```

```
while (a < 3) {  
    System.out.println(a);  
    a++;  
}
```

2 < 3 이므로 중괄호{} 내부를 실행한다.

while문 코드 분석6

- 0 1 2 출력하기

```
int a = 0;
```

```
while (a < 3) {  
    System.out.println(a);  
    a++;  
}
```

2를 출력하고 a를 1 증가시킨다.(a는 3을 가진 상태)

while문 코드 분석7

- 0 1 2 출력하기

```
int a = 0;
```

```
while (a < 3) {  
    System.out.println(a);  
    a++;  
}
```

3 < 3 이 아니므로 while문 실행을 종료한다.

do while문

- while문은 조건식에서 사용하는 변수의 초기값에 따라 반복문이 한 번도 실행되지 않을 수 있음
- do while문을 사용하면 변수의 초기값과 상관 없이 반드시 최초 1번은 반복문을 실행하게 됨
- 이외의 다른 특징은 while문과 동일함

do while문 형식

■ 형식

```
do {  
    실행문1  
    실행문2  
} while (조건식);
```

실행문1, 실행문2를 처리한 후 조건식을 실행한다.

마지막 세미콜론(;)이 누락되지 않도록 주의해야 한다.

for문

- 반복문의 또 다른 종류
- while문에 비해서 가독성(보기 좋아 코드 해석이 쉽다)이 좋다는 특징을 가짐
- 반복해서 사용할 값의 범위가 명확하거나,
반복 횟수가 정해져 있는 경우라면
while문보다 for문을 추천함

for문 형식

■ 형식

```
for (초기상태; 조건식; 상태변화) {  
    실행문1  
    실행문2  
}
```

"초기상태"는 오직 최초 한 번만 실행하고
"조건식" -> 중괄호{} 본문 -> 상태변화 순으로 반복한다.

```
for (초기상태; 조건식; 상태변화) {  
    실행문  
}
```

반복할 실행문이 하나인 경우 중괄호{}는 생략 가능하다.

for문 코드 분석1

- 0 1 2 출력하기

초기상태로 $a = 0$ 값을 가진다.

```
for (int a = 0; a < 3; a++) {  
    System.out.println(a);  
}
```

for문 코드 분석2

■ 0 1 2 출력하기

0 < 3 이므로 중괄호{}를 실행하고 0을 출력한다.

```
for (int a = 0; a < 3; a++) {  
    System.out.println(a);  
}
```

①
②

for문 코드 분석3

- 0 1 2 출력하기

a를 1 증가시킨다.(a는 1을 가진 상태)

```
for (int a = 0; a < 3; a++) {  
    System.out.println(a);  
}
```

for문 코드 분석4

■ 0 1 2 출력하기

1 < 3 이므로 중괄호{}를 실행하고 1을 출력한다.

```
for (int a = 0; a < 3; a++) {  
    System.out.println(a);  
}
```

①
②

for문 코드 분석5

- 0 1 2 출력하기

a를 1 증가시킨다.(a는 2를 가진 상태)

```
for (int a = 0; a < 3; a++) {  
    System.out.println(a);  
}
```

for문 코드 분석6

- 0 1 2 출력하기

2 < 3 이므로 중괄호{}를 실행하고 2를 출력한다.

```
for (int a = 0; a < 3; a++) {  
    System.out.println(a);  
}
```

①
②

for문 코드 분석7

- 0 1 2 출력하기

a를 1 증가시킨다.(a는 3을 가진 상태)

```
for (int a = 0; a < 3; a++) {  
    System.out.println(a);  
}
```


for문 코드 분석8

- 0 1 2 출력하기

3 < 3 이 아니므로 for문 실행을 종료한다.

```
for (int a = 0; a < 3; a++) {  
    System.out.println(a);  
}
```

반복문의 중첩

- 반복문 내부에 또 다른 반복문이 포함되어 있는 것을 중첩 반복문(Nested Loop)이라고 함
- 2가지 이상의 값이 반복적으로 사용되는 경우에는 중첩이 필요함
- 2차원 평면 형식으로 표현하는 데이터들을 처리할 때도 중첩이 필요함

반복문 중첩 예시 - Outer Loop

■ 1일차 수업

1교시
2교시
3교시

■ 2일차 수업

1교시
2교시
3교시

■ 3일차 수업

1교시
2교시
3교시

→ 바깥쪽 반복문(Outer Loop)으로 구성한다.

```
for(int day = 1; day <= 3; day++) {  
    System.out.println(day + "일차 수업");  
}
```

반복문 중첩 예시 - Inner Loop

■ 1일차 수업

1교시
2교시
3교시

■ 2일차 수업

1교시
2교시
3교시

■ 3일차 수업

1교시
2교시
3교시

→ 안쪽 반복문(Inner Loop)으로 구성한다.

```
for(int day = 1; day <= 3; day++) {  
    System.out.println(day + "일차 수업");  
    for(int hour = 1; hour <= 3; hour++) {  
        System.out.println(hour + "교시");  
    }  
}
```

```

each: function(e, t, n) {
  var r, i = 0,
      o = e.length,
      a = M(e);
  if (n) {
    if (a) {
      for (; o > i; i++)
        if (r = t.apply(e[i], n), r === !1) break;
    } else
      for (i in e)
        if (r = t.apply(e[i], n), r === !1) break;
  } else if (a) {
    for (; o > i; i++)
      if (r = t.call(e[i], e[i]), r === !1) break;
  } else
    for (i in e)
      if (r = t.call(e[i], e[i]), r === !1) break;
  return e;
},
trim: b && !b.call("\uffff\u00a0") ? function(e) {
  return null == e ? "" : b.call(e);
} : function(e) {
  return null == e ? "" : (e + "").replace(C, "");
},
makeArray: function(e, t) {
  var n = t || [];
  return null != e && (M(Object(e)) ? x.merge(n, "string"
),
isArray: function(e, t, n) {
  var r;
  if (t) {
    if (n) return m.call(t, e, n);
    for (r = t.length, r = r ? 0 > n ? Math.max(0, r + n) : n : 0; r < n; r++)
      if (n in t && t[r] === e) return r;
  }
}

```

3. 기타 제어문

break문

- 분기문에서 다룬대로 break문으로 switch문을 종료시킬 수 있음
- while문이나 for문과 같은 반복문도 break문으로 종료시킬 수 있음

- 기본 형식

```
for( 초기상태 ; 조건식 ; 상태변화 ) {  
    if(조건식) {  
        break;           조건식을 만족하면 for문 실행을 종료함  
    }  
    실행문  
}
```

break문의 동작

■ while문

```
while (조건식) {
```

```
    break;
```

```
}
```

while문을 종료한다.



■ for문

```
for( 초기상태 ; 조건식 ; 상태변화 ) {
```

```
    break;
```

```
}
```

for문을 종료한다.



무한루프와 break문

- 무한루프와 break문을 이용하면 보다 쉽게 반복문을 구성할 수 있음

```
while (true) {  
    if(조건식)  
        break;  
    실행문  
}
```

```
for ( ; ; ) {  
    if(조건식)  
        break;  
    실행문  
}
```

무한루프 구성 방식

1. while (true) {
 }
2. for (; ;) {
 }

continue문


- 반복문의 시작 지점으로 프로그램의 흐름이 이동함
- 반복문에서 제외하고 실행하고 싶은 항목이 있는 경우에 주로 사용함
- 기본 형식

```
for( 초기상태 ; 조건식 ; 상태변화 ) {  
    if(조건식) {  
        continue;   조건식을 만족하면 실행문을 제외하고 for문을 계속 실행함  
    }  
    실행문  
}
```

continue문의 동작

■ while문

```
while (조건식) {  
    continue;  
}
```



조건식으로 이동한다.

■ for문

```
for( 초기상태 ; 조건식 ; 상태변화 ) {  
    continue;  
}
```



상태변화로 이동한다.