# CH 4

조건문과 반복문
    if문
        블럭
        if-else문
        if-else if 문
        중첩 if문 - if문 안의 if문
    Switch문
        switch문의 제약 조건
    임의의 정수 만들기
    for문
        중첩 for문
        while문
        do-while문
    break문
        continue문
        이름 붙은 반복문

## 조건문과 반복문

- 조건문
  - 조건을 만족할 때만 {}를 수행(0~1번)
  - if, switch문
- 반복문
  - 조건을 만족하는 동안 {}를 수행 (0~n번)
  - for, while문
- 이 둘을 포괄하여 `제어문(flow control statement)`이라고 부른다

## if문

- 조건식이 참(true)일 때, 괄호{} 안의 문장들을 수행한다

```
if (조건식) {
  // 조건식이 참(true)일 때 수행될 문장들을 적는다.
}
```

```
str.equals("yes");  // 문자열 str의 내용이 "yes"일 때 (대소문자 구분)
str.equalsIgnoreCase("yes"); // 문자열 str의 내용이 "yes"일 때 (대소문자 구분 안 함)
```

## 블럭

- 여러 문장을 하나로 묶어 주는 것
- 블럭 안 문장들은 tap키를 이용해 들여쓰기를 함

## if-else문

- 돌 중 하나: 조건식이 참일 때와 거짓일 때로 나누어서 처리

```
if (조건식) {
  // 조건식이 참(true)일 때 수행될 문장들
} else {
  // 조건식이 거짓(false)일 때 수행될 문장
}
```

## if-else if 문

- 여러 개 중 하나: 여러 개 조건식을 포함한 조건식

```
if (조건식 1) {
    // 조건식 1 연산 결과가 참일 때 수행될 문장들
} else if (조건식 2) {
    // 조건식 2 연산 결과가 참일 때 수행될 문장들
} else if (조건식 3) {
    // 조건식 3 연산 결과가 참일 때 수행될 문장들
} else { // 마지막은 보통 else 블럭으로 끝나며, else 블럭은 생략 가능
    // 위의 어느 조건식도 만족하지 않을 때 수행될 문장들
}
```

- 처음에 초기화를 함으로써 else 블럭을 쓰지 않는 방법도 강구하기

```
char grade = 'D';
int num = 숫자;

if (num > 90) {
    grade = 'A';
} else if (num > 80) {
    grade = 'B';
} else if (num > 70) {
    grade = 'C';
/* } else {
    grade = 'D';
```

```
} */

// 초기 값을 D로 두면 else 블럭을 굳이 넣지 않아도 된다
```

## 중첩 if문 - if문 안의 if문

```
if (조건식 1) {
    // 조건식 1 연산 결과가 참일 때 수행될 문장들
    if (조건식 2) {
            // 조건식 1과 조건식 2가 모두 true일 때 수행될 문장들
    } else {
          // 조건식 1이 true이고, 조건식 2가 false일 때 수행될 문장들
    }
} else {
    // 조건식 1이 false일 때 수행되는 문장들
}
```

💡 else문 쓸 때 주의
괄호를 치지 않으면 `가까운 if문`과 연결된다

# Switch문

- 처리해야 하는 경우의 수가 많을 때 유용한 조건문
- if~else문에는 조건식이 true/false가 온다
- switch문에는 조건식이 정수나 문자열이 올 수 있다
- switch문은 항상 if문으로 바꿀 수 있으나, if문은 switch문으로 바꿀 수 없을 때도 있다

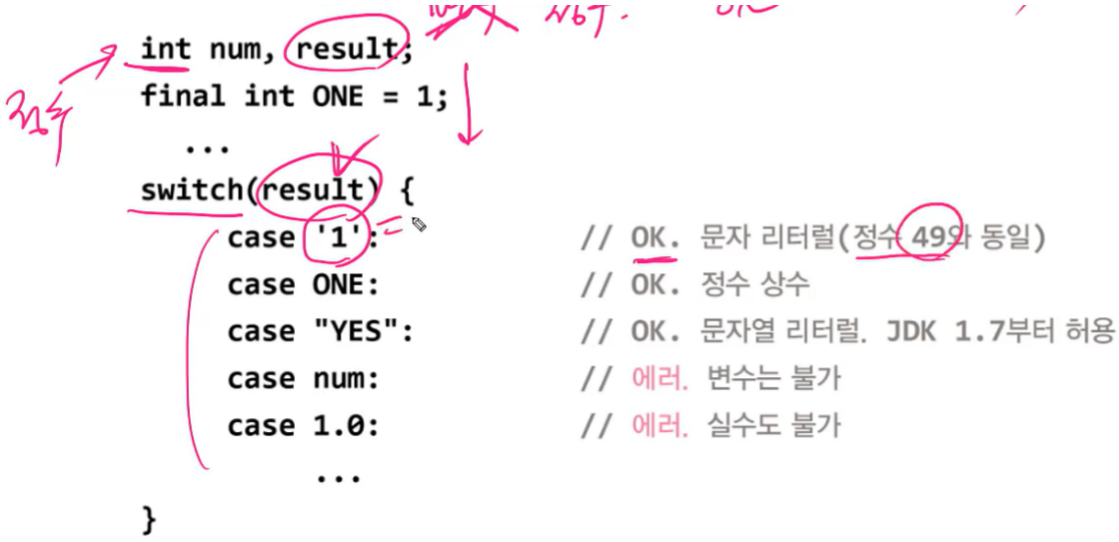
```
switch (조건식) {
      case 값1 :
            // 조건식의 결과가 값 1과 같을 경우 수행될 문장들
            break;
      case 값2 :
            // 조건식의 결과가 값 2와 같을 경우 수행될 문장들
            break; // switch문 벗어나기
      default :
            // 조건식의 결과와 일치하는 case 문이 없을 때 수행될 문장들
            // default 문은 보통 switch 문의 가장 마지막에 쓰기 때문에 break을 쓰지 않는다
}
```

1. 조건식을 계산한다
2. 조건식의 결과와 일치하는 case문으로 이동한다

3. 이후의 문장들을 수행한다
4. break문이나 switch 문의 끝을 만나면 switch 문 전체를 빠져나간다

### switch문의 제약 조건

1. switch문의 조건식 결과는 `정수` 또는 문자열(JDK 1.7부터)이어야 한다
2. case문의 값은 `정수`(변수 X), `상수`(문자 포함), 문자열만 가능하며, 중복되지 않아야 한다

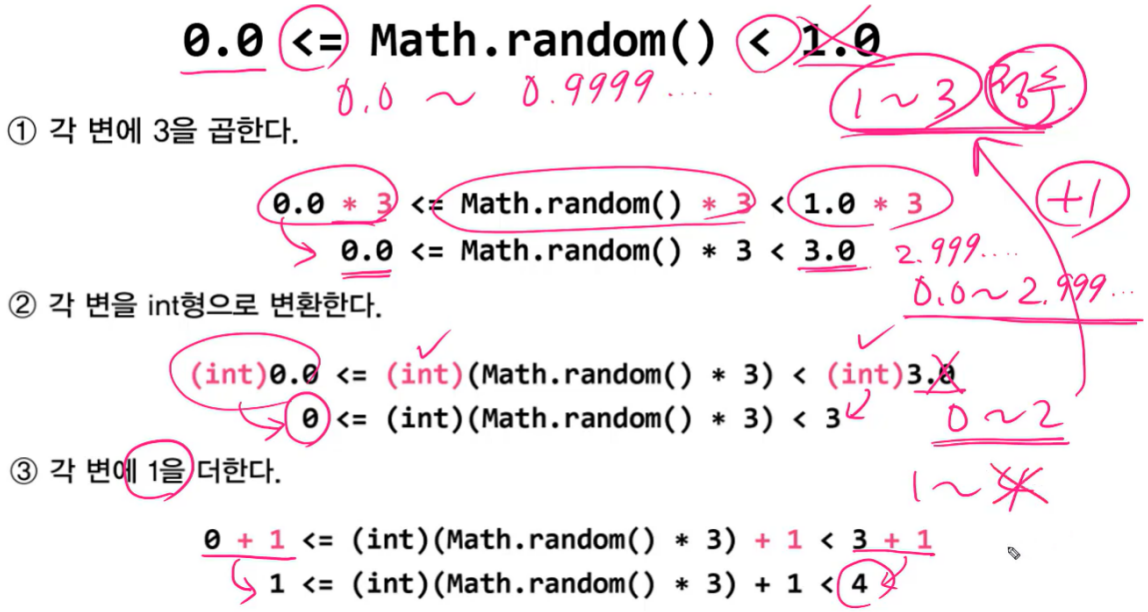


```
int num, result;
final int ONE = 1;
   ...
switch(result) {
    case '1':          // OK. 문자 리터럴(정수 49와 동일)
    case ONE:          // OK. 정수 상수
    case "YES":        // OK. 문자열 리터럴. JDK 1.7부터 허용
    case num:          // 에러. 변수는 불가
    case 1.0:          // 에러. 실수도 불가
         ...
}
```

## 임의의 정수 만들기

- 실수도 가능
- 임의의 정수: 난수
- Math.random(): 0.0과 1.0 사이 임의의 **double 값** 반환

```
0.0 <= Math.random() < 1.0
```

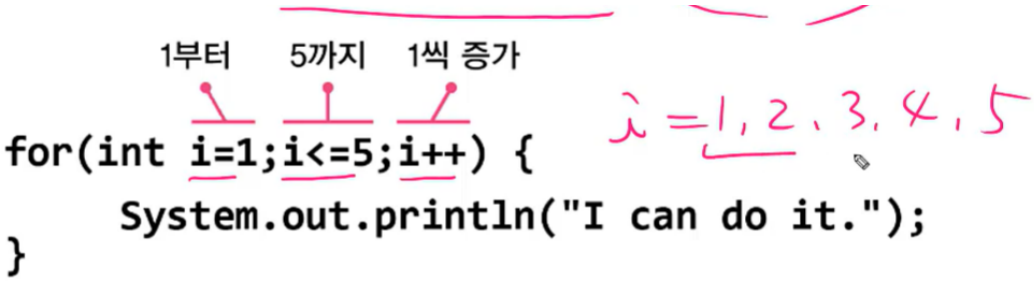


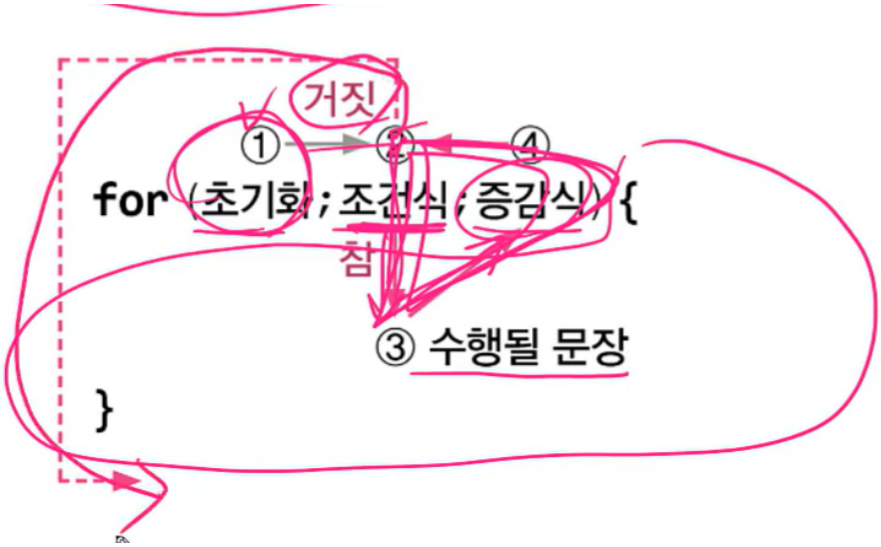

- 원하는 개별 값의 개수 곱해 주기

# for문

- 조건을 만족하는 동안 블럭{}을 반복: 반복 횟수를 알 때

cf) 반복 횟수를 모를 때는 while문이 적합






- 조건식을 생략하면 true로 간주되어서 무한 반복문이 된다

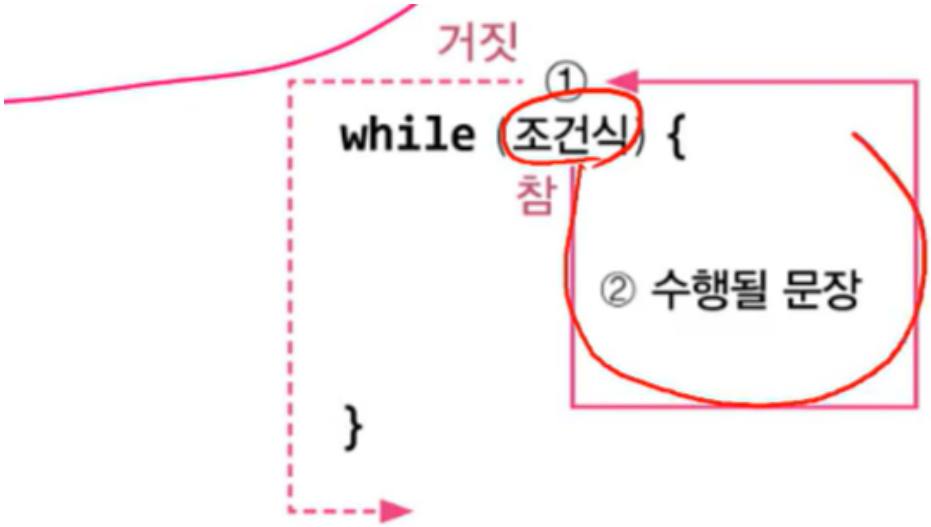
## 중첩 for문

- for문 내에 또 다른 for문을 포함시킬 수 있다

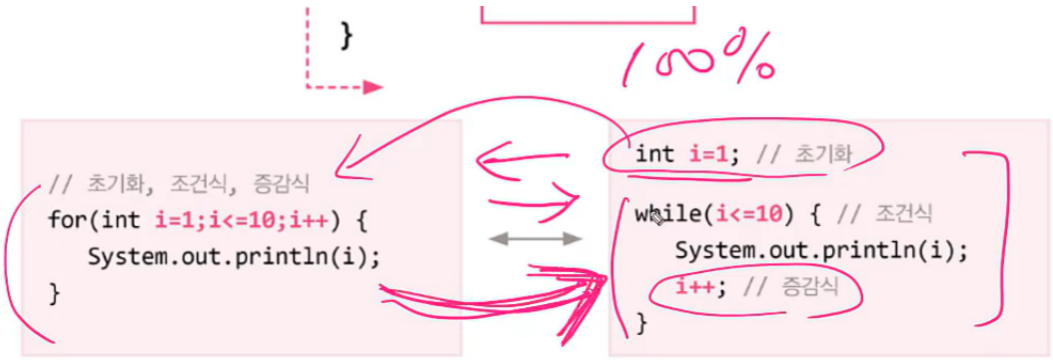
## while문

- 조건을 만족시키는 동안 블럭 {}을 반복: 반복 횟수 모를 때

```
while (조건식) {
      // 조건식의 연산 결과가 참(true)인 동안, 반복될 문장들
}
```



- while문과 for문은 상호 호환 가능하다




```
int i = 5;

    while(i-- != 0) {
      // 헷갈리면 i !=0와 i--를 나눠서 하기
      System.out.println(i + " 줄어든당.");
/* 4 줄어든당.
3 줄어든당.
2 줄어든당.
1 줄어든당.
0 줄어든당. */
```

```
    int sum = 0, i = 0;
    while (sum <= 100) {
      System.out.printf("%d - %d%n", i, sum);
      sum += ++i;
    }

/* 0 - 0
1 - 1
2 - 3
3 - 6
4 - 10
5 - 15
6 - 21
7 - 28
8 - 36
9 - 45
10 - 55
11 - 66
12 - 78
13 - 91 */
```

```
int num = 0, sum = 0;
    System.out.print("숫자를 입력하세요 : ");
    Scanner sc = new Scanner(System.in);
    String str = sc.next();
```

```
    num = Integer.parseInt(str);

    while(num != 0) {
      // num을 10으로 나눈 나머지를 sum에 더함
      sum += num % 10;
      System.out.printf("sum = %3d num = %d%n", sum, num);
      num /= 10;
    }
    System.out.println("각 자리수의 합: " + sum);
```

## do-while문

- 블럭 {}을 최소한 한 번 이상 반복: 사용자 입력받을 때 유용
- 1번에서 n번 반복

```
do {
    // 조건식의 연산 결과가 참일 때 수행될 문장들을 적는다 (처음 한 번은 무조건 실행)
} while (조건식); // 끝에 ;를 잊지 않도록 주의하기
```

```
    int num = 0, ans = 0;
    ans = (int) (Math.random() * 100) + 1; // 1과 100 사이의 숫자
    Scanner sc = new Scanner(System.in);

    do {
      System.out.print("1과 100 사이의 숫자를 입력하세요 : ");
      num = sc.nextInt();

      if (num > ans) {
        System.out.println("더 작은 숫자로 도전해 보세요.");
      } else if (num < ans) {
        System.out.println("더 큰 숫자로 도전해 보세요.");
      }
    } while (num != ans);
    System.out.println("정답입니다.");
  }
```

- while 문으로 쓰면 입력을 두 번 받아야 돼서 do-while문이 더 효율적인 코드

# break문

- 자신이 포함된 하나의 반복문을 벗어난다

```
for문의 무한 반복문: for(;;)
while문의 무한 반복문: while(true)
```

## continue문

- 자신이 포함된 반복문의 끝으로 이동: 다음 반복으로 넘어간다
- 전체 반복 중에서 특정 조건 시 반복을 건너뛸 때 유용

```
int menu = 0, num = 0;
Scanner sc = new Scanner(System.in);

while (true) {
  System.out.println("1. 불고기버거");
  System.out.println("2. 초밥");
  System.out.println("3. 파스타");
  System.out.println("1~3번 중에 메뉴를 선택하세요. (종료 : 0번)");

  String str = sc.nextLine();
  menu = Integer.parseInt(str);

  if (menu == 0) {
    System.out.println("키오스크를 종료합니다.");
    break;
  } else if (!(1 <= menu && menu <= 3)) {
    System.out.println("메뉴를 잘못 선택하셨습니다. 종료는 0번입니다.");
    continue;
  }
  System.out.println("선택하신 메뉴는 " + menu + "번입니다.");
}
```

## 이름 붙은 반복문

- 반복문에 이름을 붙여서 하나 이상의 반복문을 벗어날 수 있다
- 원래 break: 하나의 반복문을 벗어난다

예제 4-19

```
class Ex4_19
{
   public static void main(String[] args)
   {
      // for문에 Loop1이라는 이름을 붙였다.
      Loop1 : for(int i=2;i <=9;i++) {
            for(int j=1;j <=9;j++) {
               if(j==5)
                  break Loop1;
//                break;
//                continue Loop1;
//                continue;
               System.out.println(i+"*"+ j +"="+ i*j);
            } // end of for i
            System.out.println();
      } // end of Loop1

   }
}
```

결과

```
2*1=2
2*2=4
2*3=6
2*4=8
```