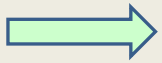
# 제 5 장
# 프로그램 흐름제어-2부
# 반복문

# 반복명령문

- 프로그램의 특정 부분을 반복적으로 실행할 필요가 있을 경우
- 대량의 데이터를 기반으로 동일 동작을 반복해야 하는 경우
  - 1부터 10000까지 합(혹은 곱)을 구하시오
  - 15!을 구하시오
  - 객프를 수강하는 각 학생의 평균점수를 구하시오 등...

# 반복문의 필요성

```
int main(void) {
int  x = 1;
int sum = 0;

sum = sum + x ;          // sum = 0+1
x = x + 1;
sum = sum + x ;          // sum = 1+2
x = x + 1;
sum = sum + x ;          // sum = 3+3
x = x + 1;
sum = sum + x ;          // sum = 6+4
x = x + 1;
sum = sum + x ;          // sum = 10+5
x = x + 1;
sum = sum + x ;          // sum = 15+6
x = x + 1;
sum = sum + x ;          // sum = 21+7
x = x + 1;
sum = sum + x ;          // sum = 28+8
x = x + 1;
sum = sum + x ;          // sum = 36+9
x = x + 1;
sum = sum + x ;          // sum = 45+10
x = x + 1;
printf("1부터 10까지 합은 %d 입니다\n", sum);
return  0;
}
```



```
int main(void) {

int x ;
int sum = 0;

for(x=1; x<11; x++)
   sum = sum+x;
}
```

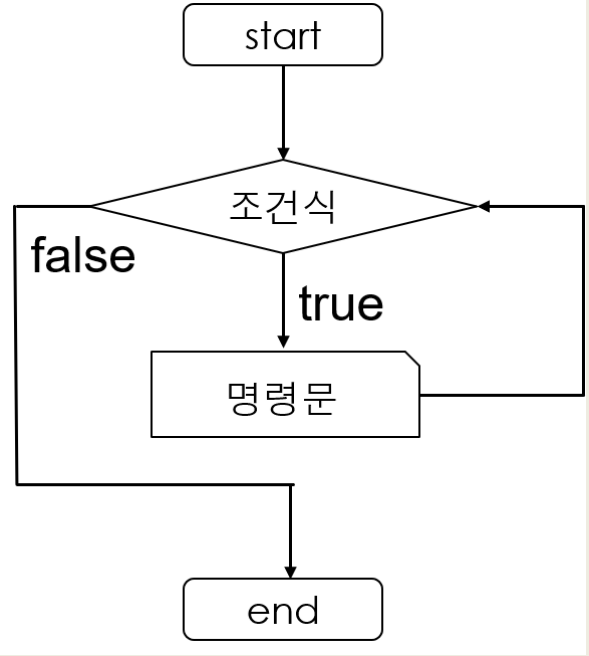
# 반복문의 종류

- while(조건식) { .....}
- for(초기조건; 반복수행조건;반복후동작) {
  ..... }
- do {..... } while(조건식)
- For-each라고 하는 특별한 반복문
  - 배열 데이터에 대해서 주로 사용

# While 명령문

- 주어진 조건이 만족되는 동안 명령문들을 반복 실행한다.

```
while( 조건식 ) {
        명령문;
}
```



```
class WhileTest {
    public static void main(String[] args){
        int count = 1;
        while(count < 11) {
            System.out.println("Count is: " + count);
            count++;
        }
    }
}
```

```
while(true) {
    // 명령문
}
```

# 특정 수에 대한 구구단 출력 예

```
class GuGuDanTest {
    public static void main(String[] args) {
        Scanner guGudan = new Scanner(System.in);
        int gugu;
        int  i = 1;
        System.out.println("구구단 중에서 출력하고 싶은 단을 입력하시오: ");
        gugu  =  guGudan.nextInt();

        while(i <= 9) {
            System.out.println(gugu + '*' + i + '=' + gugu*i);
            i++;
        }
    }
}
```

```
구구단 중에서 출력하고 싶은 단을 입력하시오: 9
9*1 = 9
9*2 = 18
9*3 = 27
....
9*9 = 81
```

# 1부터 10까지에 대한 제곱 예

```
class SquareTest {
    public static void main(String[] args) {
        System.out.println("======================");
        System.out.println("    n        n의 제곱 ₩n");
        System.out.println("======================");

        int n =1;
        while(n <= 10) {
            System.out.println(n + '\t' + n*n );
            n++;
        }
    }
}
```

```
==================
  n      n의 제곱
==================
  1       1
  2       4
  3       9
  4      16
  5      25
  6      36
  7      49
  8      64
  9      81
 10     100
```

# 1부터 n까지의 합

```
class SumTest {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int  i = 1;
        int sum = 0;
        System.out.println("정수를 입력하시오: ");
        int n = sc.nextInt();

        while(i <= n) {
            sum += i;
            i++;
        }
        System.out.println("1부터 " + n + "까지의 합은 " + sum + " 입니다");
        i++;
    }
}
```

```
정수를 입력하시오: 3
1부터 3까지의 합은 6입니다
```

# 지정 개수의 입력 값의 합

```
class SumTest {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int  i = 0;
        int sum = 0;
        int n = 0;
        while(i < 5) {
            System.out.println("값을 입력하시오: ");
            n  =  sc.nextInt();
            sum += n;
            i++;
        }
        System.out.println("입력 갓들의 합은 " + sum + " 입니다");
    }
}
```

```
값을 입력하시오: 10
값을 입력하시오: 20
값을 입력하시오: 30
값을 입력하시오: 40
값을 입력하시오: 50
합계는 150입니다.
```

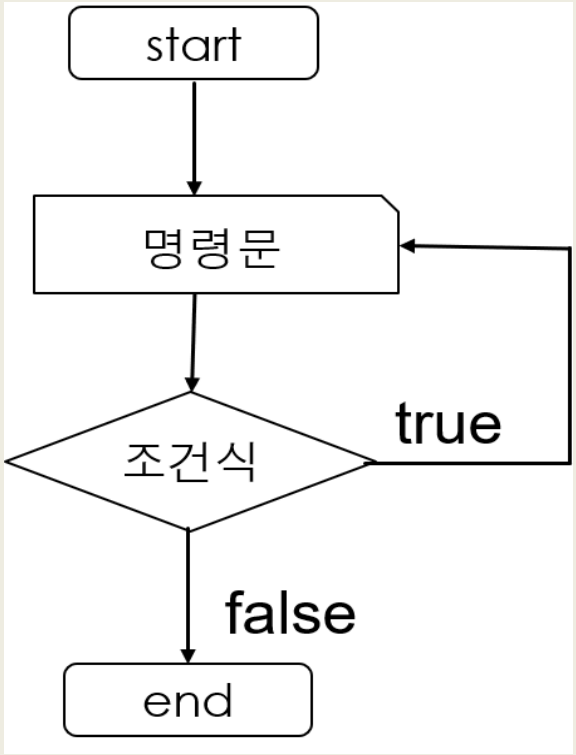
# do-While 명령문

- 반복 조건을 루프의 끝에서 검사
- 최소 한번은 명령문 실행

```
do {
    ..명령문 블록..
} while(조건식)
```

❶

❷




❶

- 무조건 최소 한번은 실행

❷

- 조건식이 true이면 반복, false이면 반복 종료
- 조건식이 없으며 컴파일 오류

```
class DoWhileTest {
    public static void main(String[] args){
        int count = 1;
        do {
            System.out.println("Count is: " + count);
            count++;
        } while(count < 11)
    }
}
```

# 글자 추측 게임

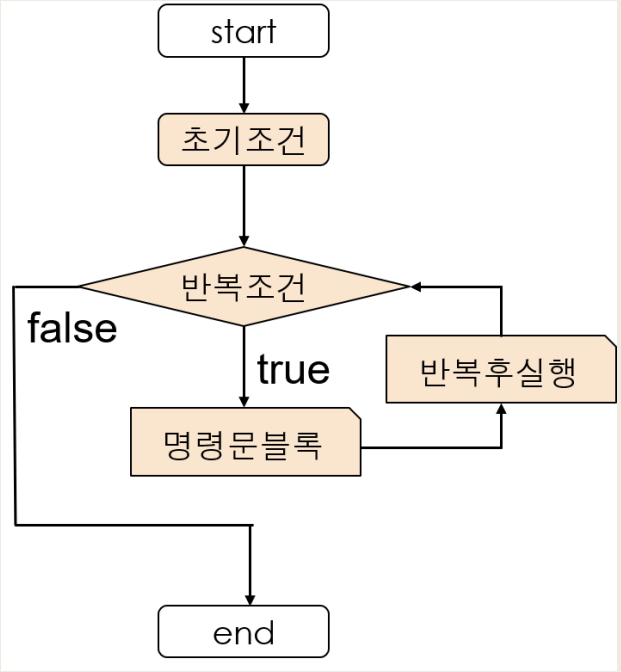
```
class GuessCharGame {

    public static void main(String[] args){
     char answer = 'm'
      char guess;
      int tries = 0;
      Scanner ch = new Scanner(System.in);
         do {
             System.out.println("문자를 추측하여 보시오: ");
             guess = ch.next();
             tries++;
              if( guess > answer )
                      System.out.println("제시한 문자의 아스키 코드 값이 높습니다.₩n");
              if( guess < answer )
                       System.out.println("제시한 문자의 아스키 코드값이 낮습니다.₩n");
         } while(guess != answer );
          System.out.println("축하합니다. 시도횟수=%d ₩n", tries);
    }
}
```

```
정답을 추측하여 보시오: a
제시한 문자의 아스키 코드값이 낮습니다.
정답을 추측하여 보시오: s
제시한 문자의 아스키 코드값이 높습니다.
정답을 추측하여 보시오: b
제시한 문자의 아스키 코드값이 낮습니다.
정답을 추측하여 보시오: z
제시한 문자의 아스키 코드값이 높습니다.
정답을 추측하여 보시오: m
축하합니다. 시도횟수=5
```

# for 명령문

- 정해진 횟수만큼 반복하는 구조
  - 특정 범위 내에서 원하는 **횟수**만큼 반복할 수 있는 아주 효율적인 방법

```
for (초기조건 ; 반복조건 ; 반복후 실행) {
        명령문 블록;
}
```



- for 명령문 형식

```
for (초기조건 ; 반복조건 ; 반복후 실행) {
        명령문 블록;
}
```

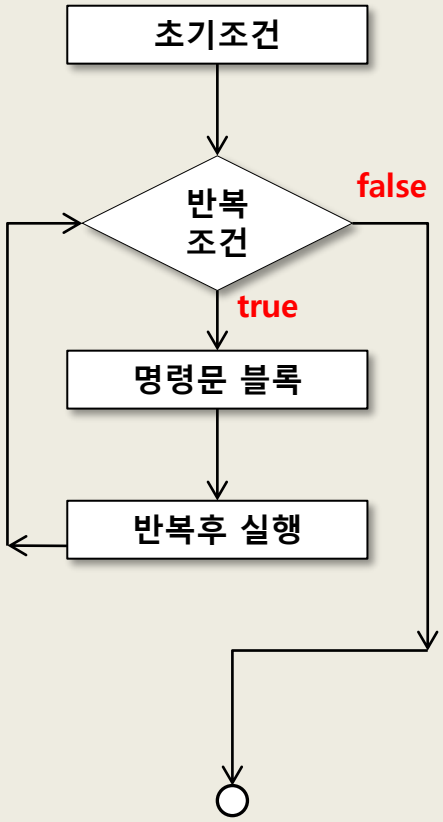
  - 초기조건은 반복실행프를 시작할 때 단 1회만 동작하는 초기화 동작이다.
  - 반복조건은 for 블록을 1회 실행 종료하고, 반복후 실행이 완료된 후, 조건 테스트 결과 true로 판명나면, for 블록을 다시 실행하고, false로 판명되면 반복은 종료되며 for 블록을 탈출한다.
  - 반복후 실행은 for 블록이 한번 수행된 후 실행되며, 보편적으로 사용되는 동작은 어떤 변수 값에 대한 증감 동작이지만, 그렇지 않을 수도 있다

❶ ❷ ❹

```
for(초기조건; 반복조건; 반복후실행)
{
    ..명령문 블록..
}
```

❸

for 키워드




❶

- for 문이 실행한 후 오직 한번만 실행되는 초기화 작업
- 콤마(',')로 구분하여 여러 문장 나열 가능
- 초기할 일이 없으면 비어둘 수 있음

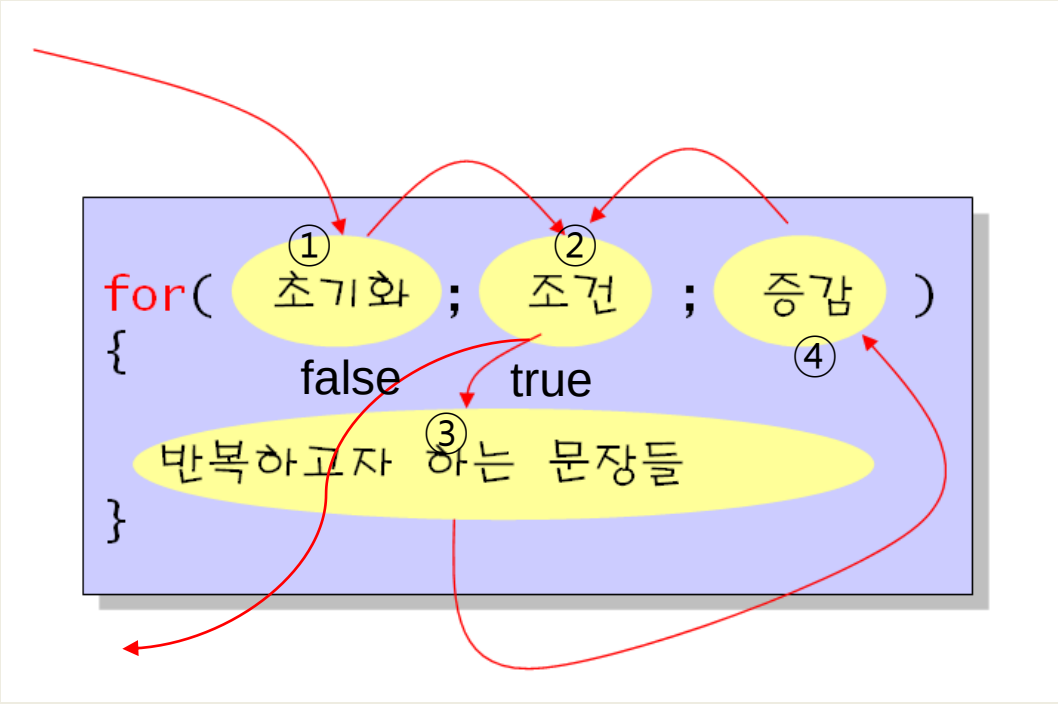
❷

- 논리형 변수나 논리 연산만 가능
- 반복 조건이 true이면 반복 계속, false이면 반복 종료후 for-문 탈출
- 반복 조건이 true 상수인 경우, 무한 반복
- 반복 조건이 비어 있으면 true로 간주

❸

- 반복 수행할 작업 명령들
- 콤마(',')로 구분하여 여러 문장 나열 가능

① 초기화식을 실행
② 반복 조건을 나타내는 조건식을 테스트
③ 조건식의 값이 false면 for-명령문 실행을 종료하고 for-loop 탈출
true면 명령문 실행
④ 반복후 실행(보통 증감 동작)하고 ②로 재반복



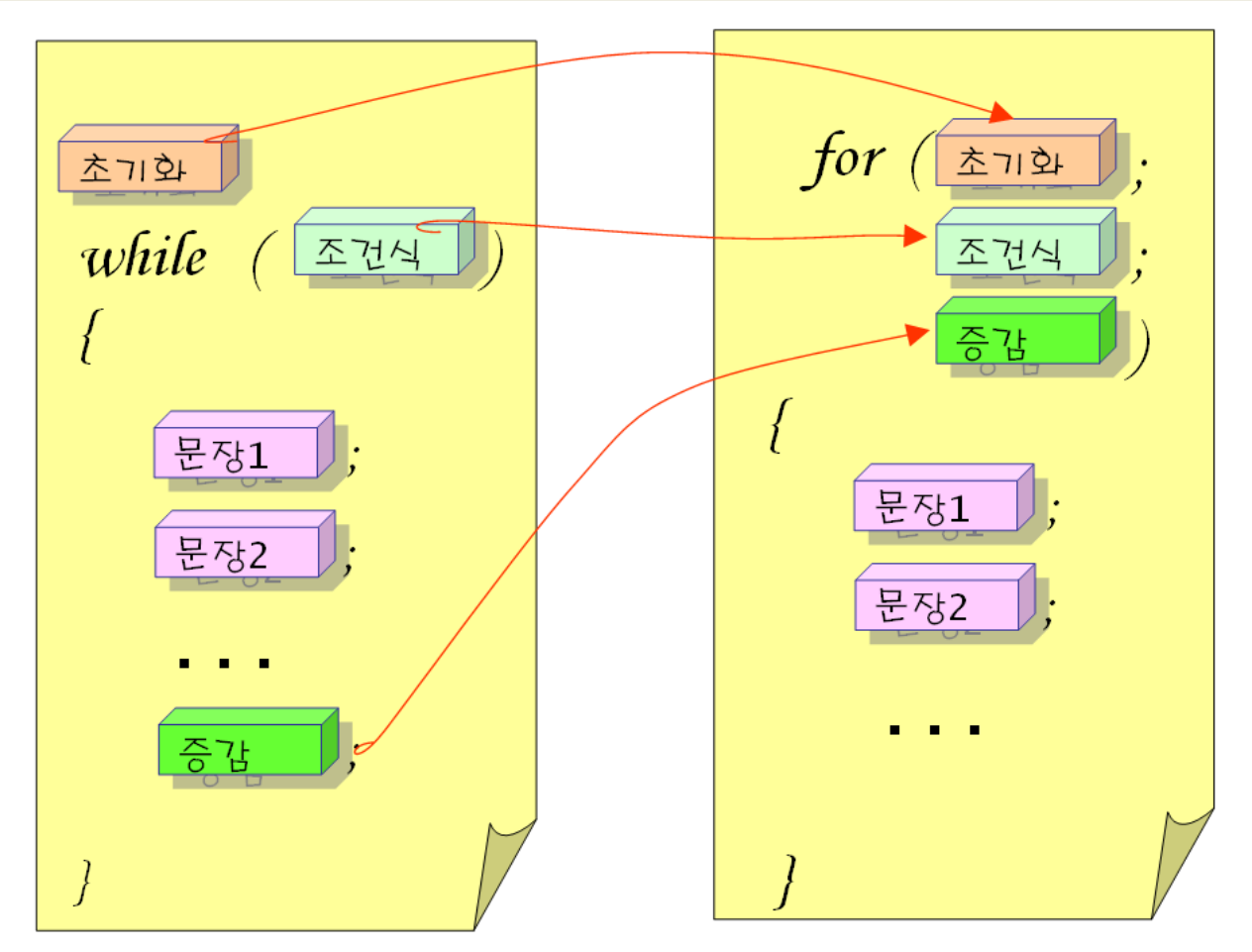
```
class ForTest {
    public static void main(String[] args){
        for(int i=1; i<11; i++) {
            System.out.println("Count is: " + i);
        }
    }
}
```

```
Count is: 1
Count is: 2
Count is: 3
Count is: 4
Count is: 5
Count is: 6
Count is: 7
Count is: 8
Count is: 9
Count is: 10
```

```
class FactorialTest {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        long factorial = 1;
        int n;
        System.out.println("정수를 입력하시요:");
        n = sc.nextInt();
        for(int i=1; i<=n; i++) {
            factorial *= i;
        }
        System.out.println(n + "! = " factorial + "입니다 ");
    }
}
```

```
정수를 입력하시요: 10
10! = 3628800입니다.
```

# While-문 vs. for-문 변환



```
for (int i = 10; i > 0; i-- )
    printf("Hello World!₩n");
```

뺄셈 사용

```
for (int i = 0; i < 10; i += 2 )
    printf("Hello World!₩n");
```

2씩 증가

```
for (int i = 1; i < 10; i *= 2 )
    printf("Hello World!₩n");
```

2를 곱한다.

```
for (int i = 0; i < 100; i = (i * i) + 2 )
    printf("Hello World!₩n");
```

어떤 수식이라도 가능

```
for  ( ; i<100; i++ )
    printf("Hello World!₩n");
```

한부분이 없을 수도 있다.

```
for  ( ; ; )
    printf("Hello World!₩n");
```

무한반복

# 중첩 반복문(nested loop)

- 중첩 반복문(nested loop): 반복문 안에 다른 반복문이 위치
- 이론적으로는 몇 번이고 중첩 반복 가능
- 너무 많은 중첩 반복은 프로그램 구조를 복잡하게 하므로 보통 2중 또는 3중 반복 정도가 적당

```
for(int i = 0; i < 5; i++)
        {
                for(int j = 0; j < 10; j++)
                        System.out.print("*");
                System.out.println("");
        }
```

```
**********
**********
**********
**********
**********
```

## 구구단 출력

2중 중첩된 for문을 사용하여 구구단을 출력하는 프로그램을 작성하시오.
한 줄에 한 단씩 출력한다.

```
public class NestedLoop {
   public static void main (String[] args) {
      int i, j;

      for (i = 1;i < 10; i++,System.out.println()) {
         for (j = 1;j < 10; j++,System.out.print('\t')) {
            System.out.print(i + "*" + j + "=" + i*j);
         }
      }
   }
}
```

```
1*1=1   1*2=2   1*3=3   1*4=4   1*5=5   1*6=6   1*7=7   1*8=8   1*9=9
2*1=2   2*2=4   2*3=6   2*4=8   2*5=10  2*6=12  2*7=14  2*8=16  2*9=18
3*1=3   3*2=6   3*3=9   3*4=12  3*5=15  3*6=18  3*7=21  3*8=24  3*9=27
4*1=4   4*2=8   4*3=12  4*4=16  4*5=20  4*6=24  4*7=28  4*8=32  4*9=36
5*1=5   5*2=10  5*3=15  5*4=20  5*5=25  5*6=30  5*7=35  5*8=40  5*9=45
6*1=6   6*2=12  6*3=18  6*4=24  6*5=30  6*6=36  6*7=42  6*8=48  6*9=54
7*1=7   7*2=14  7*3=21  7*4=28  7*5=35  7*6=42  7*7=49  7*8=56  7*9=63
8*1=8   8*2=16  8*3=24  8*4=32  8*5=40  8*6=48  8*7=56  8*8=64  8*9=72
9*1=9   9*2=18  9*3=27  9*4=36  9*5=45  9*6=54  9*7=63  9*8=72  9*9=81
```

# 분기 명령문

- break 문, labelled break
- continue 문, labelled continue
- return 문

# break 문

- break 문 특정 블록을 탈출하고자 하는 경우 사용
  - Switch문, 반복문 및 중괄호로 둘러싸인 블록문 등...
- 단순 break와 labelled break
- 단순 break문일 경우
  - 중첩된 블록: 내부 블록만 탈출, 외부 블록은 탈출하지 못함
- Labelled break 문을 사용하면 중첩된 블록을 한번에 탈출 가능
- 2중 중첩된 for-문을 LABEL을 이용해서 한번에 탈출하는 예

```
LABEL:
for (초기 조건; 반복 조건; 반복 후 실행) {
        for (초기 조건; 반복 조건; 반복 후 실행) {
                ..............
                break LABEL;
                ..............
        }
}
 ..............
```

# 단순 break 문 예

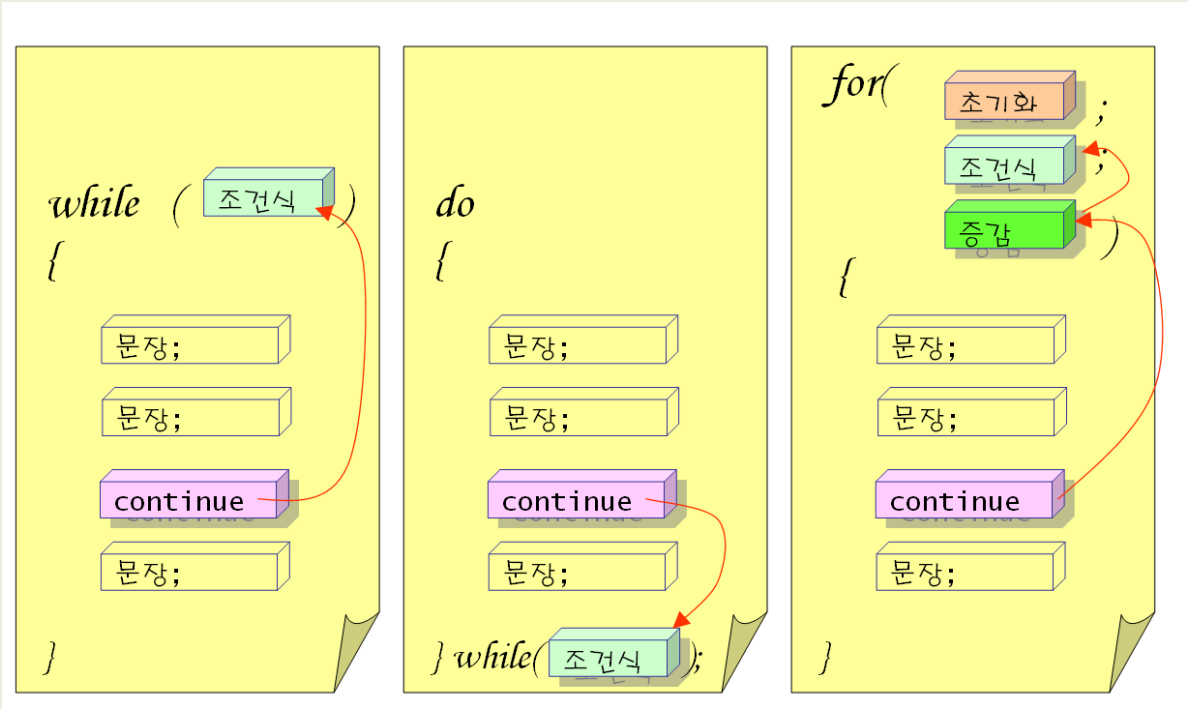
```
import java.util.Scanner;
public class BreakExample {
   public static void main (String[] args) {
      Scanner in = new Scanner(System.in);
      int num = 0;

      while (true) {
         if (in.nextInt() == -1)
            break;
         num++;
      }
      System.out.println("입력된 숫자 개수는 " + num);
   }
}
```

```
10
8
9
5
-1
입력된 숫자 개수는 4
```

# continue 문

- 현재의 반복문장수행을 중단하고 다음차례 반복을 새로시작한다
  - 반복문을 빠져 나가지 않으면서 특정 값에 대해 반복문 내의 실행문을 건너 뛸 때 사용
  - 즉, continue 후반부는 실행하지 않고 다시 반복을 시작
- 단순 continue 문과 labelled continue 문이 있다

단순 continue



- Labelled continue 문

```
LABEL:
for (초기 작업; 반복 조건; 반복 후 작업) {
        for (초기 작업; 반복 조건;반복 후 작업)
        {
            ..............
            continue LABEL;
            ..............
        }
}
```

# 1부터 100까지 짝수들의 합

```
import java.util.Scanner;

public class SumOfEvenNumbers {

        public static void main(String[] args) {

                int sum = 0;

                for(int i = 0; i<100; i++) {
                   if(i%2 == 1)
                      continue;
                   sum += i;
                }
                System.out.println("1부터 100까지 짝수의 합 = " + sum);
        }
}
```

```
1부터 100까지의 합 = 245
```

# 소문자를 대문자로 변환

```
import java.util.Scanner;

public class ToUpperCase {

        public static void main(String[] args) {

                short letter;
                Scanner sc = new Scanner(System.in);
                while(true) {
                   System.out.println("영문 소문자를 입력하세요: ");
                   letter = sc.nextShort();
                   if(letter == 'Q');
                      break;              // 작업종료
                   it(letter < 'a' || letter > 'z')
                      continue;           // 입력오류
                   letter  = letter – 32;
                   System.out.println("변환된 소문자는 " + letter + "입니다");
                }
        }
}
```

```
소문자를 입력하시오: a
변환된 대문자는 A입니다.
소문자를 입력하시오: b
변환된 대문자는 B입니다.
소문자를 입력하시오: c
변환된 대문자는 C입니다.
소문자를 입력하시오: Q
```

# return 문

- return 명령문은 현재 수행 중인 메소드를 종료하고 프로그램 실행 제어를 메소드를 호출했던 위치로 넘기는 명령문
- return 명령문은 2가지 유형을 가지고 있는데
  - 하나는 값(리턴값)을 넘기는 형태이고,
    - 특정 값을 리턴하기 위해서는 단순히 return 다음에 넘겨줄 값 혹은 그 값을 연산하는 수식을 다음과 같이 적어주면 된다.
    - **return ++count;**
  - 다른 하나는 다음곽 같이 단순히 실행만 넘기는 형태이다
    - return;.