5. 반복문

#1.인강/0.자바/1.자바-입문

반복문 시작

반복문은 이름 그대로 특정 코드를 반복해서 실행할 때 사용한다.

자바는 다음 3가지 종류의 반복문을 제공한다.

`while`, `do-while`, `for`

먼저 간단한 예제를 통해 반복문이 왜 필요한지 이유를 알아보자.

1을 한 번씩 더해서 총 3번 더하는 간단한 코드를 만들어보자.

While1_1

```
package loop;

public class While1_1 {

    public static void main(String[] args) {
        int count = 0;

        count = count + 1;
        System.out.println("현재 숫자는:" + count);
        count = count + 1;
```

```
            System.out.println("현재 숫자는:" + count);
            count = count + 1;
            System.out.println("현재 숫자는:" + count);
        }
    }
```

출력

```
현재 숫자는:1
현재 숫자는:2
현재 숫자는:3
```

단순히 `count`에 값을 1씩 3번 더하는 단순한 예제이다. 최종 결과는 `3`이다.

`count = count + 1`은 증감 연산자(`++`)를 사용해서 다음과 같이 개선할 수 있다.

```
//개선
count++;
System.out.println("현재 숫자는:" + count);
count++;
System.out.println("현재 숫자는:" + count);
count++;
System.out.println("현재 숫자는:" + count);
```

하지만 같은 코드가 3번 반복되고 있다. 이번에는 1을 한 번씩 더해서 총 100번 더하는 코드를 만들어보자
아마도 직접 작성한다면 같은 코드가 100번 반복될 것이다.
이렇게 특정 코드를 반복해서 실행할 때 사용하는 것이 바로 반복문이다.
반복문에는 `while`, `for`문이 있다. 먼저 `while`문부터 알아보자.

while문1

while문은 조건에 따라 코드를 반복해서 실행할 때 사용한다.

```
while (조건식) {
    // 코드
}
```

- 조건식을 확인한다. 참이면 코드 블럭을 실행하고, 거짓이면 while문을 벗어난다.
- 조건식이 참이면 코드 블럭을 실행한다. 이후에 코드 블럭이 끝나면 다시 조건식 검사로 돌아가서 조건식을 검사한다.(무한 반복)

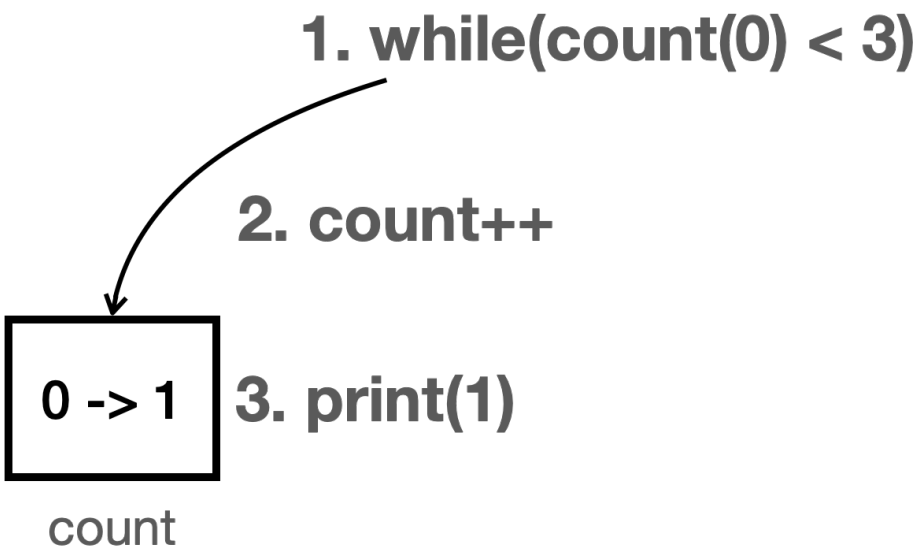
while문을 사용해서 1을 한 번씩 더해서 총 3번 더하는 코드를 만들어보자

While1_2

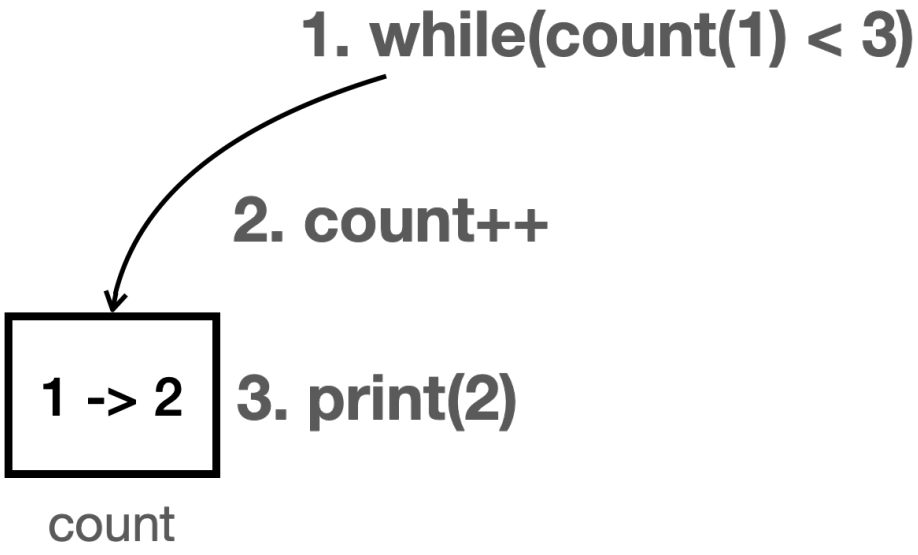
```
package loop;

public class While1_2 {

    public static void main(String[] args) {
        int count = 0;

        while (count < 3) {
            count++;
            System.out.println("현재 숫자는:" + count);
        }

    }
}
```

출력 결과

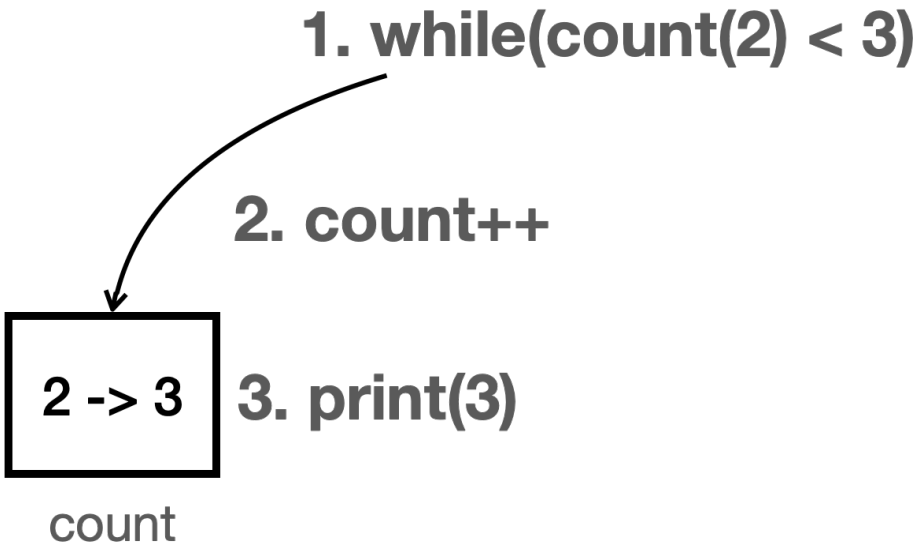
```
현재 숫자는:1
현재 숫자는:2
현재 숫자는:3
```



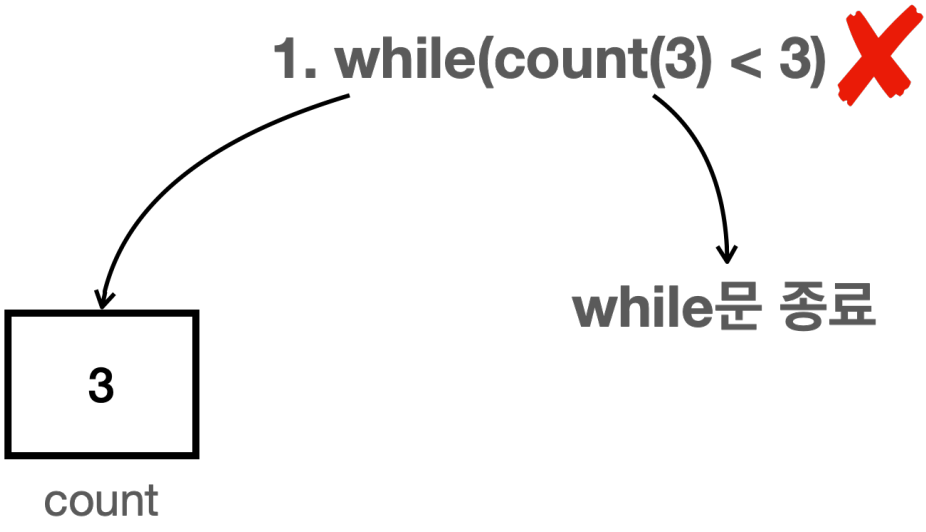
while문 실행1, count=0



while문 실행2, count=1



while문 실행3, count=2



while문 실행4, count=3

`while (count < 3)`에서 코드 블럭을 반복 실행한다. 여기서 `count`의 값이 `1`,`2`,`3`으로 점점 커지다가 결국 `count < 3`이 거짓이 되면서 `while`문을 빠져나간다.

`while(count < 3)`에 있는 숫자를 `while(count < 100)`으로 변경하면 `while`문의 코드 블럭을 100번 반복한다.

while문2

이번에는 난이도를 조금 높여보자. 다음 문제를 같이 풀어보자.

문제: 1부터 하나씩 증가하는 수를 3번 더해라 (1 ~ 3 더하기)

이 문제는 1부터 하나씩 증가하는 수이기 때문에 1 + 2 + 3을 더해야 한다.

우선 `while`문을 사용하지 않고 단순 무식하게 풀어보자.

While2_1

```
package loop;

public class While2_1 {
    public static void main(String[] args) {
        int sum = 0;

        sum = sum + 1; //sum(0) + 1 -> sum(1)
        System.out.println("i=" + 1 + " sum=" + sum);

        sum = sum + 2; //sum(1) + 2 -> sum(3)
        System.out.println("i=" + 2 + " sum=" + sum);

        sum = sum + 3; //sum(3) + 3 -> sum(6)
        System.out.println("i=" + 3 + " sum=" + sum);
    }
}
```

출력 결과

```
i=1 sum=1
i=2 sum=3
```

```
i=3 sum=6
```

이 코드의 정답은 맞다. 하지만 개선할 점이 많이 있는데, 무엇보다 변경에 유연하지 않다.

다음과 같이 요구사항이 변경되었다.

문제: 10부터 하나씩 증가하는 수를 3번 더해라 (10 ~12더하기)

이렇게 되면 `10 + 11 + 12`를 계산 해야한다. 문제는 코드를 너무 많이 변경해야 한다는 점이다.

변수를 사용해서 더 변경하기 쉬운 코드로 만들어보자. 변경되는 부분을 변수 `i`로 바꾸어보자.

문제: i부터 하나씩 증가하는 수를 3번 더해라 (i ~ i+2더하기)

While2_2

```
package loop;

public class While2_2 {
    public static void main(String[] args) {
        int sum = 0;
        int i = 1;

        sum = sum + i; //sum(0) + i(1) -> sum(1)
        System.out.println("i=" + i + " sum=" + sum);
        i++; //i=2

        sum = sum + i; //sum(1) + i(2) -> sum(3)
        System.out.println("i=" + i + " sum=" + sum);
        i++; //i=3

        sum = sum + i; //sum(3) + i(3) -> sum(6)
        System.out.println("i=" + i + " sum=" + sum);
    }
}
```

출력 결과

```
//i=1
i=1 sum=1
i=2 sum=3
i=3 sum=6
```

변수 `i`를 사용한 덕분에 `i`의 값만 변경하면 나머지 코드를 전혀 변경하지 않아도 된다.

`i=10`으로 변경하면 다른 코드의 변경 없이 앞서 이야기한 `10 + 11 + 12`의 문제도 바로 풀 수 있다.

출력 결과

```
//i=10
i=10 sum=10
i=11 sum=21
i=12 sum=33
```

좋은 코드인지 아닌지는 변경 사항이 발생했을 때 알 수 있다. 변경 사항이 발생했을 때 변경해야 하는 부분이 적을수록 좋은 코드이다.

지금까지 변수를 사용해서 하나의 문제를 잘 해결했다. 이번에는 새로운 변경사항이 등장했다.
기존 문제: i부터 하나씩 증가하는 수를 3번까지 더해라 (i ~ i+2 더하기)
새로운 문제: i부터 하나씩 증가하는 수를 endNum(마지막 수)까지 더해라 (i ~ endNum 더하기)

예)

- `i=1`, `endNum=3` 이라고 하면 1 ~ 3까지 총 3번 더해야한다.
- `i=1`, `endNum=10` 이라고 하면 1 ~ 10까지 총 10번 더해야한다.
- `i=10`, `endNum=12` 이라고 하면 10 ~ 12까지 총 3번 더해야한다.

먼저 `i=1`, `endNum=3` 이라고 생각하고 단순하게 문제를 풀어보자.

While2_3

```java
package loop;

public class While2_3 {
    public static void main(String[] args) {
        int sum = 0;
        int i = 1;
        int endNum = 3;

        sum = sum + i;
        System.out.println("i=" + i + " sum=" + sum);
        i++;

        sum = sum + i;
        System.out.println("i=" + i + " sum=" + sum);
        i++;

        sum = sum + i;
        System.out.println("i=" + i + " sum=" + sum);
        i++;
    }
```

```
}
```

실행 결과

```
i=1 sum=1
i=2 sum=3
i=3 sum=6
```

`i=1`, `endNum=3` 이므로 다음 코드를 총 3번 반복해야 한다.

```
sum = sum + i;
System.out.println("i=" + i + " sum=" + sum);
i++;
```

그런데 `i=1`, `endNum=10` 와 같이 변경하면 이 코드를 총 10번 반복해야 한다. 따라서 같은 코드를 더 많이 추가해야 한다.

이 문제를 제대로 풀기 위해서는 **코드가 실행되는 횟수를 유연하게 변경**할 수 있어야 한다. 한마디로 같은 코드를 반복 실행할 수 있어야 한다.

`while` 문을 사용하면 원하는 횟수 만큼 같은 코드를 반복 실행할 수 있다.

While2_3 - 코드 변경

```
package loop;

public class While2_3 {
    public static void main(String[] args) {
        int sum = 0;
        int i = 1;
        int endNum = 3;

        while (i <= endNum) {
            sum = sum + i;
            System.out.println("i=" + i + " sum=" + sum);
            i++;
        }
    }
}
```

반복 횟수 정하기

`i` 가 `endNum` 이 될때 까지 반복해서 코드를 실행하면 된다.

- `i=1`, `endNum=3` 이라면 3번 반복하면 된다. `i=1 -> 2 -> 3`

- `i=3`, `endNum=4`라면 2번 반복하면 된다. `i=3 -> 4`

while문 작성하기

- `while`문에서 `i <= endNum` 조건을 통해 `i`가 `endNum`이 될 때 까지 코드 블럭을 실행한다.
- `i`가 `endNum`보다 크면 `while`문을 종료한다.

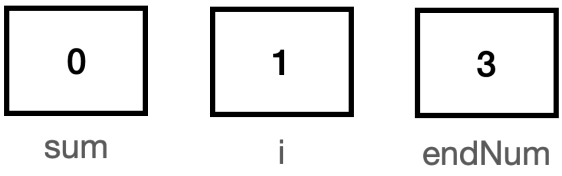
실행 결과

```
//i=1, endNum=3
i=1 sum=1
i=2 sum=3
i=3 sum=6

//i=1, endNum=10
i=1 sum=1
i=2 sum=3
i=3 sum=6
i=4 sum=10
i=5 sum=15
i=6 sum=21
i=7 sum=28
i=8 sum=36
i=9 sum=45
i=10 sum=55

//i=10, endNum=12
i=10 sum=10
i=11 sum=21
i=12 sum=33
```

그림을 통해 코드를 분석해보자.

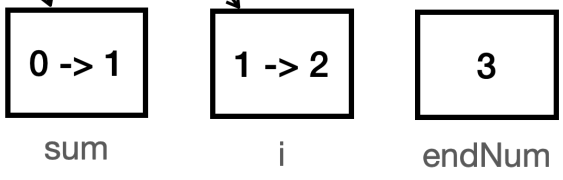
왼쪽은 `while`의 조건식을 체크하는 단계이고, 오른쪽은 조건식을 통과하고 나서 `while`문의 코드 블럭을 실행하는 부분이다.

```
while(i(1) <= endNum(3)) {
        sum = sum + i
        print(i=?, sum=?)
        i++
}
```




```
while(i(1) <= endNum(3)) {
        sum = sum(0) + i(1)
        print(i=1, sum=1)
        i++
}
```

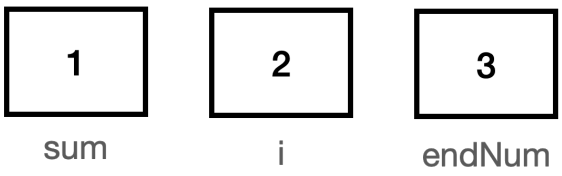
1 2



i=1, endNum=3

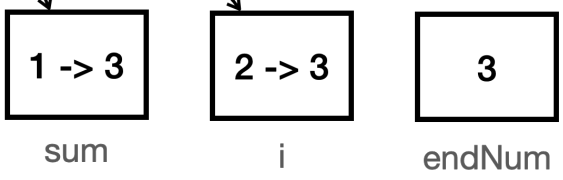
- 조건식을 만족한다.
- `i=1, sum=1` 을 출력한다.

```
while(i(2) <= endNum(3)) {
        sum = sum + i
        print(i=?, sum=?)
        i++
}
```



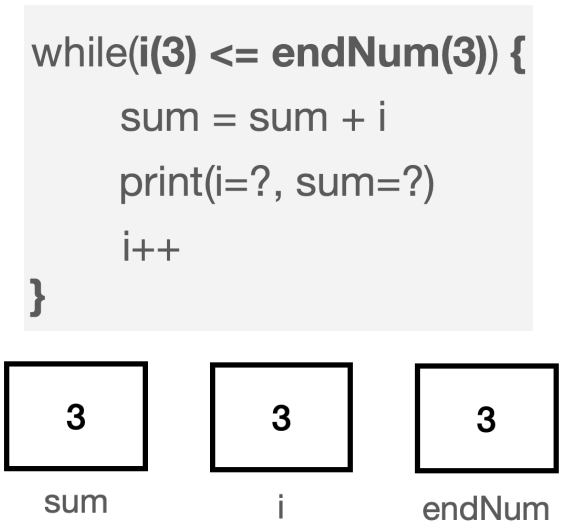

```
while(i(2) <= endNum(3)) {
        sum = sum(1) + i(2)
        print(i=2, sum=3)
        i++
}
```

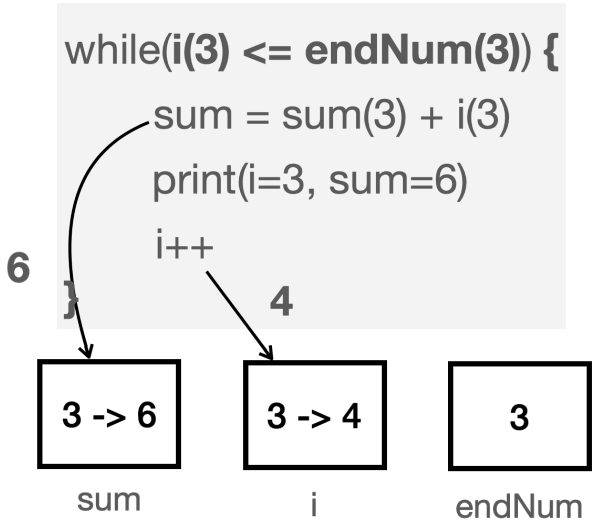
3 3



i=2, endNum=3

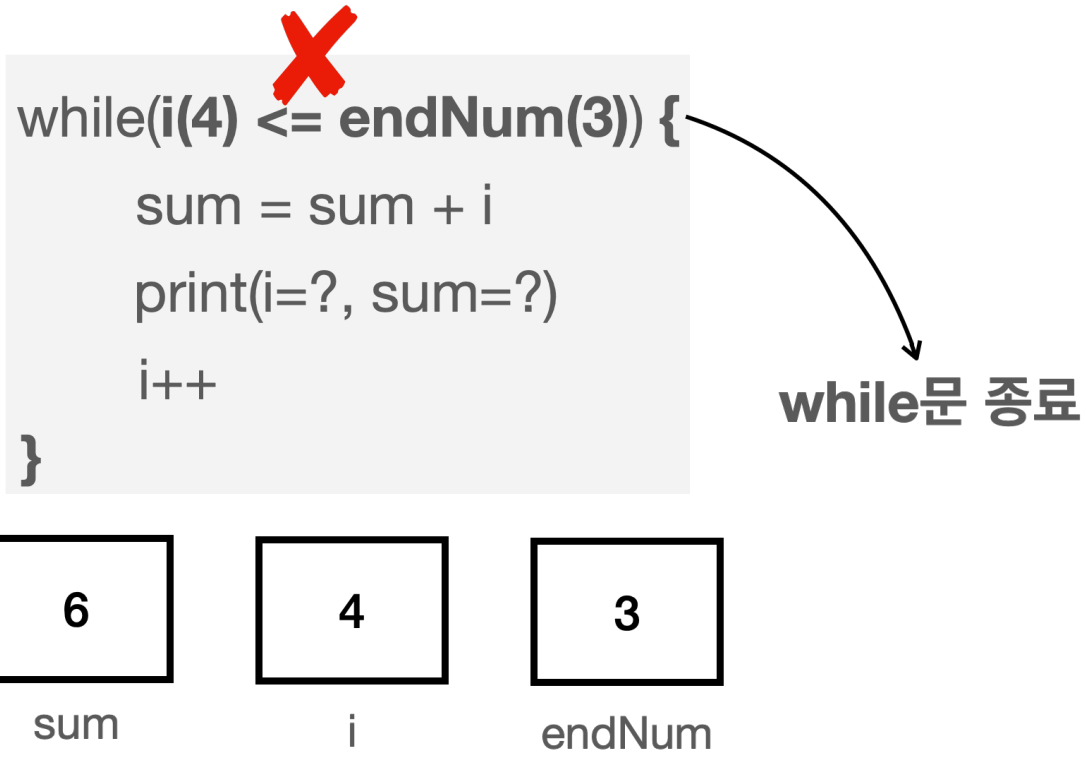
- 조건식을 만족한다.
- `i=2, sum=3` 을 출력한다.





i=3, endNum=3

- 조건식을 만족한다.
- `i=3, sum=6`을 출력한다.



i=4, endNum=3

- 조건식을 만족하지 않는다.
- `while`문을 종료한다.

실행 코드 분석

```
sum(0), i(1), endNum(3)
//루프 1
while (i(1) <= endNum(3)) -> true
sum(0)+i(1) -> sum(1)
```

```
i(1)++ -> i(2)

//루프 2
while (i(2) <= endNum(3)) -> true
sum(1)+i(2) -> sum(3)
i(2)++ -> i(3)

//루프 3
while (i(3) <= endNum(3)) -> true
sum(3)+i(3) -> sum(6)
i(3)++ -> i(4)

//루프 4
while (i(4) <= endNum(3)) -> false
```

do-while문

`do-while` 문은 `while` 문과 비슷하지만, 조건에 상관없이 무조건 한 번은 코드를 실행한다.

do-while문 구조

```
do {
    // 코드
} while (조건식);
```

예를 들어서 조건에 만족하지 않아도 한 번은 현재 값을 출력하고 싶다고 하자.
먼저 `while` 문을 사용한 예제를 보자

DoWhile1

```
package loop;

public class DoWhile1 {

    public static void main(String[] args) {
        int i = 10;
        while (i < 3) {
            System.out.println("현재 숫자는:" + i);
            i++;
```

```
        }
    }
}
```

`i=10`이기 때문에 `while (i < 3)` 조건식은 거짓이 된다. 따라서 아무것도 출력되지 않는다.

출력 결과

```
//없음
```

이번에는 `do-while`문을 사용해보자.

DoWhile2

```
package loop;

public class DoWhile2 {

    public static void main(String[] args) {
        int i = 10;
        do {
            System.out.println("현재 숫자는:" + i);
            i++;
        } while (i < 3);

    }
}
```

`do-while`문은 최초 한번은 항상 실행된다. 따라서 먼저 `현재 숫자는:10`이 출력된다.
코드 블럭을 실행 후에 조건식을 검증하는데, `i=10`이기 때문에 `while (i < 3)` 조건식은 거짓이 된다. 따라서 `do-while`문을 빠져나온다.

출력 결과

```
현재 숫자는:10
```

`do-while`문은 최초 한번은 코드 블럭을 꼭 실행해야 하는 경우에 사용하면 된다.

break, continue

`break`와 `continue`는 반복문에서 사용할 수 있는 키워드다.

`break`는 반복문을 즉시 종료하고 나간다. `continue`는 반복문의 나머지 부분을 건너뛰고 다음 반복으로 진행하는데 사용된다.
참고로 `while`, `do-while`, `for`와 같은 모든 반복문에서 사용할 수 있다.

break

```
while(조건식) {
  코드1;
  break;//즉시 while문 종료로 이동한다.
  코드2;
}
//while문 종료
```

`break`를 만나면 `코드2`가 실행되지 않고 while문이 종료된다.

continue

```
while(조건식) {
  코드1;
  continue;//즉시 조건식으로 이동한다.
  코드2;
}
```

`continue`를 만나면 `코드2`가 실행되지 않고 다시 조건식으로 이동한다. 조건식이 참이면 `while`문을 실행한다.

예제를 통해서 알아보자.

문제: 1부터 시작해서 숫자를 계속 누적해서 더하다가 합계가 10보다 처음으로 큰 값은 얼마인가?

1 + 2 + 3 ... 계속 더하다가 처음으로 합이 10보다 큰 경우를 찾으면 된다.

Break1

```
package loop;

public class Break1 {
    public static void main(String[] args) {
        int sum = 0;
        int i = 1;

        while (true) {
            sum += i;
            if (sum > 10) {
                System.out.println("합이 10보다 크면 종료: i=" + i + " sum=" + sum);
                break;
            }
```

```
            i++;
        }
    }
}
```

- 조건식을 잘 보면 `true`라고 되어있다. 조건이 항상 참이기 때문에 이렇게 두면 `while`문은 무한 반복된다. 물론 `break`문이 있기 때문에 중간에 빠져나올 수 있다.
- 만약 `sum > 10` 조건을 만족하면 결과를 출력하고, `break`를 사용해서 `while`문을 빠져나간다.

실행 결과

```
합이 10보다 크면 종료: i=5 sum=15
```

문제: 1부터 5까지 숫자를 출력하는데, 숫자가 3일 때는 출력을 건너뛰어야 한다.

Continue1

```
package loop;

public class Continue1 {
    public static void main(String[] args) {
        int i = 1;

        while (i <= 5) {
            if (i == 3) {
                i++;
                continue;
            }
            System.out.println(i);
            i++;
        }
    }
}
```

`i==3`인 경우 `i`를 하나 증가하고 `continue`를 실행한다. 따라서 이 경우에는 `i`를 출력하지 않고 바로 `while (i <= 5)` 조건식으로 이동한다.

실행 결과

```
1
2
4
5
```

실행 결과를 보면 3일 때는 출력하지 않은 것을 확인할 수 있다.

for문1

for문도 while문과 같은 반복문이고, 코드를 반복 실행하는 역할을 한다. for문은 주로 반복 횟수가 정해져 있을 때 사용한다.

for문 구조

```
for (1.초기식; 2.조건식; 4.증감식) {
    // 3.코드
}
```

for문은 다음 순서대로 실행된다.

- 1. 초기식이 실행된다. 주로 반복 횟수와 관련된 변수를 선언하고 초기화 할 때 사용한다. 초기식은 딱 1번 사용된다.
- 2. 조건식을 검증한다. 참이면 코드를 실행하고, 거짓이면 for문을 빠져나간다.
- 3. 코드를 실행한다.
- 4. 코드가 종료되면 증감식을 실행한다. 주로 초기식에 넣은 반복 횟수와 관련된 변수의 값을 증가할 때 사용한다.
- 5. 다시 2. 조건식 부터 시작한다. (무한 반복)

for문은 복잡해 보이지만 while문을 조금 더 편하게 다룰 수 있도록 구조화 한 것 뿐이다.

예를 들어 1부터 10까지 출력하는 for문은 다음과 같다.

```
for (int i = 1; i <= 10; i++) {
    System.out.println(i);
}
```

- 1. 초기식이 실행된다. `int i = 1`
- 2. 조건식을 검증한다. `i <= 10`
- 3. 조건식이 참이면 코드를 실행한다. `System.out.println(i);`
- 4. 코드가 종료되면 증감식을 실행한다. `i++`
- 5. 다시 2. 조건식을 검증한다. (무한 반복) 이후 `i <= 10` 조건이 거짓이 되면 for문을 빠져나간다.

For1

```
package loop;
```

```
public class For1 {

    public static void main(String[] args) {
        for (int i = 1; i <= 10; i++) {
            System.out.println(i);
        }
    }
}
```

문제: i부터 하나씩 증가하는 수를 endNum(마지막 수)까지 더해라 (i ~ endNum 더하기)

for문을 사용해서 풀어보자

For2

```
package loop;

public class For2 {
    public static void main(String[] args) {
        int sum = 0;
        int endNum = 3;

        for (int i = 1; i <= endNum; i++) {
            sum = sum + i;
            System.out.println("i=" + i + " sum=" + sum);
        }
    }
}
```

출력 결과

```
i=1 sum=1
i=2 sum=3
i=3 sum=6
```

for vs while

앞서 같은 문제를 풀었던 while문과 for문을 서로 비교해보자.

While2_3

```
package loop;
```

```java
public class While2_3 {
    public static void main(String[] args) {
        int sum = 0;
        int i = 1;
        int endNum = 3;

        while (i <= endNum) {
            sum = sum + i;
            System.out.println("i=" + i + " sum=" + sum);
            i++;
        }
    }
}
```

둘을 비교했을 때 for문이 더 깔끔하다는 느낌을 받을 것이다. for문은 초기화, 조건 검사, 반복 후 작업 등이 규칙적으로 한 줄에 모두 들어 있어 코드를 이해하기 더 쉽다. 특히 반복을 위해 값이 증가하는 카운터 변수를 다른 부분과 명확하게 구분할 수 있다.

`for (int i = 1; i <= endNum; i++)`
여기서는 바로 변수 `i`가 카운터 변수이다. 증가하면서 반복 횟수가 올라가고, 또 변수 `i`를 사용해서 계속 반복할지 아니면 빠져나갈지 판단할 수 있다.
이렇게 반복 횟수에 직접적인 영향을 주는 변수를 선언부터, 값 증가, 또 조건식에 활용까지 `for (초기식; 조건식; 증감식)` 구조를 활용해서 처리하는 것이다.
덕분에 개발자는 루프 횟수와 관련된 코드와 나머지 코드를 명확하게 구분할 수 있다.
반면에 while을 보면 변수 `i`를 선언하는 부분 그리고 `i++`로 증가하는 부분이 기존 코드에 분산되어 있다.

for문2

for문 구조

```java
for (초기식; 조건식; 증감식) {
    // 코드
}
```

for문에서 초기식, 조건식, 증감식은 선택이다. 다음과 같이 모두 생략해도 된다. 단 생략해도 각 영역을 구분하는 세미콜론(`;`)은 유지해야 한다.

```
for (;;) {
    // 코드
}
```

이렇게 하면 조건이 없기 때문에 무한 반복하는 코드가 된다. 따라서 다음과 같은 코드가 된다.

```
while (true) {
    // 코드
}
```

for문을 사용해서 다음 문제를 풀어보자.

문제: 1부터 시작하여 숫자를 계속 누적해서 더하다가 합계가 10보다 큰 처음 값은 얼마인가?

1 + 2 + 3 ... 계속 더하다가 처음으로 합이 10보다 큰 경우를 찾으면 된다.

Break2

```
package loop;

public class Break2 {
    public static void main(String[] args) {
        int sum = 0;
        int i = 1;

        for (; ; ) {
            sum += i;
            if (sum > 10) {
                System.out.println("합이 10보다 크면 종료: i=" + i + " sum=" + sum);
                break;
            }
            i++;
        }
    }
}
```

- `for (; ; )`를 보면 조건식이 없다. for문은 조건이 없으면 무한 반복한다.
- `sum > 10`조건을 만족하면 `break`를 사용해서 while문을 빠져나간다.

실행 결과

```
합이 10보다 크면 종료: i=5 sum=15
```

for문은 증가하는 값이 무엇인지 초기식과 증감식을 통해서 쉽게 확인할 수 있다. 이 코드나 while문을 보면 어떤 값이 반복에 사용되는 증가 값인지 즉시 확인하기는 어렵다.

여기서는 `i`가 증가하는 값이다. 따라서 다음과 같이 `i`를 `for`문에 넣어서 관리하도록 변경하면 더 깔끔한 코드가 된다.

Break3

```
package loop;

public class Break3 {
    public static void main(String[] args) {
        int sum = 0;

        for (int i = 1; ; i++) {
            sum += i;
            if (sum > 10) {
                System.out.println("합이 10보다 크면 종료: i=" + i + " sum=" + sum);
                break;
            }
        }
    }
}
```

정리하면 for문이 없이 while문으로 모든 반복을 다를 수 있다. 하지만 카운터 변수가 명확하거나, 반복 횟수가 정해진 경우에는 for문을 사용하는 것이 구조적으로 더 깔끔하고, 유지보수 하기 좋다.

> **참고**
> for문을 좀 더 편리하게 사용하도록 도와주는 향상된 for문 또는 for-each문으로 불리는 반복문도 있다. 이 부분은 뒤에서 설명한다.

중첩 반복문

반복문은 내부에 또 반복문을 만들 수 있다. `for`, `while` 모두 가능하다.

다음 코드를 작성하고 실행해보자.

Nested1

```
package loop;

public class Nested1 {
```

```
    public static void main(String[] args) {
        for (int i = 0; i < 2; i++) {
            System.out.println("외부 for 시작 i:" + i);
            for (int j = 0; j < 3; j++) {
                System.out.println("-> 내부 for " + i + "-" + j);
            }
            System.out.println("외부 for 종료 i:" + i);
            System.out.println(); //라인 구분을 위해 실행
        }
    }
}
```

실행 결과

```
외부 for 시작 i:0
-> 내부 for 0-0
-> 내부 for 0-1
-> 내부 for 0-2
외부 for 종료 i:0

외부 for 시작 i:1
-> 내부 for 1-0
-> 내부 for 1-1
-> 내부 for 1-2
외부 for 종료 i:1
```

외부 for는 2번, 내부 for는 3번 실행된다. 그런데 외부 for 1번당 내부 for가 3번 실행되기 때문에 외부(2) * 내부(3) 해서 총 6번의 내부 for 코드가 수행된다.

문제와 풀이1

> **코딩이 처음이라면 필독!**
> 프로그래밍이 처음이라면 아직 코딩 자체가 익숙하지 않기 때문에 문제와 풀이에 상당히 많은 시간을 쓰게 될 수 있다. 강의를 들을 때는 다 이해가 되는 것 같았는데, 막상 혼자 생각해서 코딩을 하려니 잘 안되는 것이다. 이것은 아직 코딩이 익숙하지 않기 때문인데, 처음 코딩을 하는 사람이라면 누구나 겪는 자연스러운 현상이다.
> 문제를 스스로 풀기 어려운 경우, 너무 고민하기 보다는 먼저 **강의 영상의 문제 풀이 과정을 코드로 따라하면서 이해하자. 반드시 코드로 따라해야 한다.** 그래야 코딩하는 것에 조금씩 익숙해질 수 있다. 그런 다음에 정답을 지우

> 고 스스로 문제를 풀어보면 된다. 참고로 강의를 듣는 시간만큼 문제와 풀이에도 많은 시간을 들어야 제대로 성장할 수 있다!

문제: 자연수 출력

처음 10개의 자연수를 출력하는 프로그램을 작성해 보세요. 이때, `count` 라는 변수를 사용해야 합니다.
while문, for문 2가지 버전의 정답을 만들어야 합니다.

출력 예시:

```
1
2
3
4
5
6
7
8
9
10
```

해답: 자연수 출력 - while

```
package loop.ex;

public class WhileEx1 {

    public static void main(String[] args) {
        int count = 1;
        while (count <= 10) {
            System.out.println(count);
            count++;
        }
    }
}
```

해답: 자연수 출력 - for

```
package loop.ex;

public class ForEx1 {

    public static void main(String[] args) {
        for (int count = 1; count <= 10; count++) {
```

```
            System.out.println(count);
        }
    }
}
```

문제: 짝수 출력

반복문을 사용하여 처음 10개의 짝수를 출력하는 프로그램을 작성해 보세요. 이때, `num`이라는 변수를 사용하여 수를 표현해야 합니다.
while문, for문 2가지 버전의 정답을 만들어야 합니다.

출력 예시:

```
2
4
6
8
10
12
14
16
18
20
```

해답: 짝수 출력 - while

```
package loop.ex;

public class WhileEx2 {

    public static void main(String[] args) {
        int num = 2;
        int count = 1;
        while (count <= 10) {
            System.out.println(num);
            num += 2;
            count++;
        }
    }
}
```

해답: 짝수 출력 - for

```java
package loop.ex;

public class ForEx2 {

    public static void main(String[] args) {
        for (int num = 2, count = 1; count <= 10; num += 2, count++) {
            System.out.println(num);
        }
    }
}
```

문제: 누적 합 계산

반복문을 사용하여 1부터 `max`까지의 합을 계산하고 출력하는 프로그램을 작성해 보세요. 이때, `sum`이라는 변수를 사용하여 누적 합을 표현하고, `i`라는 변수를 사용하여 카운트(1부터 max까지 증가하는 변수)를 수행해야 합니다. while문, for문 2가지 버전의 정답을 만들어야 합니다.

출력 예시:

```
//max=1
1

//max=2
3

//max=3
6

//max=100
5050
```

정답: 누적 합 계산 - while

```java
package loop.ex;

public class WhileEx3 {

    public static void main(String[] args) {
        int max = 100;

        int sum = 0;
        int i = 1;
```

```
        while (i <= max) {
            sum += i;
            i++;
        }
        System.out.println(sum);
    }
}
```

정답: 누적 합 계산 - for

```
package loop.ex;

public class ForEx3 {

    public static void main(String[] args) {
        int max = 100;

        int sum = 0;
        for (int i = 1; i <= max; i++) {
            sum += i;
        }
        System.out.println(sum);
    }
}
```

문제와 풀이2

문제: 구구단 출력

중첩 for문을 사용해서 구구단을 완성해라.

출력 형태

```
1 * 1 = 1
1 * 2 = 2
...
9 * 9 = 81
```

정답: 구구단 출력

```
package loop.ex;

public class NestedEx1 {
    public static void main(String[] args) {
        for(int i = 1; i <= 9; i++) {
            for(int j = 1; j <= 9; j++) {
                System.out.println(i + " * " + j + " = " + i * j);
            }
        }
    }
}
```

문제: 피라미드 출력

`int rows`를 선언해라.
이 수만큼 다음과 같은 피라미드를 출력하면 된다.

참고: `println()`은 출력후 다음 라인으로 넘어간다. 라인을 넘기지 않고 출력하려면 `print()`를 사용하면 된다.
예) `System.out.print("*")`

출력 형태

```
//rows = 2
*
**

//rows = 5
*
**
***
****
*****
```

정답: 피라미드 출력

```
package loop.ex;

public class NestedEx2 {
```

```
    public static void main(String[] args) {
        int rows = 5;

        for(int i = 1; i <= rows; i++) {
            for(int j = 1; j <= i; j++) {
                System.out.print("*");
            }
            System.out.println();
        }
    }
}
```

정리

while vs for

for문

장점:

1. 초기화, 조건 체크, 반복 후의 작업을 한 줄에서 처리할 수 있어 편리하다.
2. 정해진 횟수만큼의 반복을 수행하는 경우에 사용하기 적합하다.
3. 루프 변수의 범위가 for 루프 블록에 제한되므로, 다른 곳에서 이 변수를 실수로 변경할 가능성이 적다.

단점:

1. 루프의 조건이 루프 내부에서 변경되는 경우, for 루프는 관리하기 어렵다.
2. 복잡한 조건을 가진 반복문을 작성하기에는 while문이 더 적합할 수 있다.

while문

장점:

1. 루프의 조건이 루프 내부에서 변경되는 경우, while 루프는 이를 관리하기 쉽다.
2. for 루프보다 더 복잡한 조건과 시나리오에 적합하다.
3. 조건이 충족되는 동안 계속해서 루프를 실행하며, 종료 시점을 명확하게 알 수 없는 경우에 유용하다.

단점:

1. 초기화, 조건 체크, 반복 후의 작업이 분산되어 있어 코드를 이해하거나 작성하기 어려울 수 있다.
2. 루프 변수가 while 블록 바깥에서도 접근 가능하므로, 이 변수를 실수로 변경하는 상황이 발생할 수 있다.

한줄로 정리하자면 정해진 횟수만큼 반복을 수행해야 하면 for문을 사용하고 그렇지 않으면 while문을 사용하면 된다.
물론 이것이 항상 정답은 아니니 기준으로 삼는 정도로 이해하자