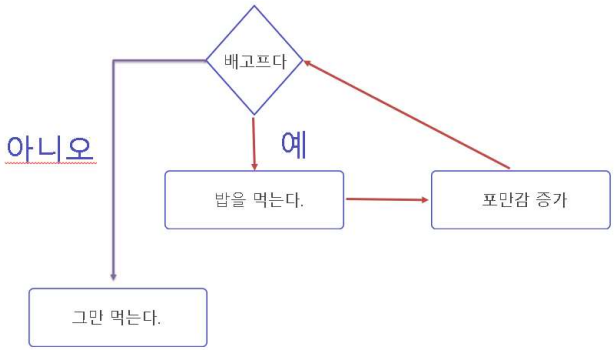
JAVA 4강 요약

강사 박주병

1.반복문

반복문이란 프로그램을 만들다보면 같은 혹은 비슷한 코드들을 계속 적어줘야 할 때가 있다. 가령 구구단을 출력하는 프로그램을 생각해 보자면 단수 x 정수 = 결과 로 단수만 변경되면서 계속 같은 것을 출력하는 것이다. 이를 직접 println을 이용해서 적어주는 것은 효율적이지 못하며 더 많은 구구단을 출력하는 것도 불가능하다.
그래서 특정 조건을 만족하는 동안 같은 코드를 계속 반복적으로 실행해주는 기능이 필요하다.

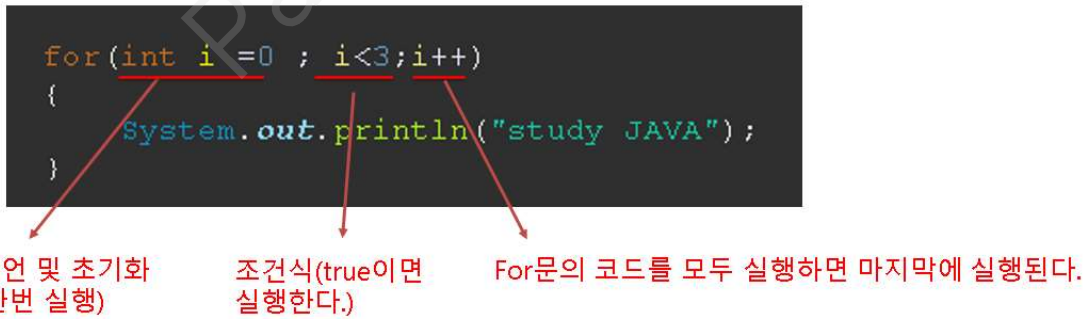


이러한 반복문에는 크게 for 과 while 두가지가 존재한다.

2.for문

```
for(초기화 ; 조건식 ; 증감식)
{
  반복되어질 코드
}
```

for문의 기본구성은 위와 같다. for문은 크게 4개의 영역으로 나눠 볼 수 있다. 초기화, 조건식, 증감식, for문의 내용 이제 하나씩 자세히 살펴 보도록 하자.



for문의 실행순서

```
①          ②      ④
for(int i =0 ; i<3;i++)
{
  ③ System.out.println("study JAVA");
}
```

1번 i번수를 선언하고 0으로 값을 초기화 한다.
2번 초기화 한 i변수가 3보다 작은지 판단한다 -> 0이므로 true 가 된다.
3번 2번에서 조건식이 true이면 3번으로 넘어와 for문 내부의 코드를 실행한다.
4번 for문 내부에 코드가 모두 실행되고 '}'를 만나면 4번으로 실행흐름이 넘어간다. i의 값을 1 증가 시킨다.
2번 1번은 최초 진입시 딱 한번만 실행되고 4번에서 증감식을 수행한후 에는 2번으로 넘어가서 조건식을 수행한다.
현재 i 값은 1이므로 여전히 true가 되어 3번으로 넘어간다.
3번 위의 내용과 동일하다. 코드 실행이 끝나면 4번으로 넘어간다.
4번 다시 I값을 증가 시킨다. 이제 I값은 2 이다.
2번 I값은 2 이므로 여전히 true 이다. 따라서 3번으로 간다
3번 코드가 실행되고 4번으로 넘어 간다.
4번 I의 값을 3으로 증가 시키고 2번으로 넘어 간다.
2번 I가 3이므로 false가 되어 3번이 실행되지 않고 for문을 종료 한다. 그리고 I는 for문 내부에서 선언된 지역변수 이므로 같이 소멸된다.

2.1 초기화

아래그림은 i 라는 변수를 int 타입으로 선언하여 0으로 초기화 하고 있는 코드 이다.

```
for(int i =0
{
```

지금까지 사용해왔던 변수의 선언과 초기화와 크게 다르지 않다. 단지 해당 변수는 for문의 지역변수로 for문의 영역을(for문 중괄호 영역) 벗어나면 소멸된다.

다음은 초기화 부분의 다양한 사용방법 들이다.

```
for(int i =0,j=13 ; i<3;i
{
    System.out.println("s
}
```

여러 개의 변수를 선언하고 초기화도 가능하다
(권장하지 않음, 반복을 결정하는 변수만 선언하고 그 외에 필요하다면 for문 외부에서 따로 선언하는게 일반적)

```
int num= 5;
for(num=1 ; num<3;num++)
{
   System.out.println("study JAVA");
}
```

For문 외부에서 선언된 변수를 활용해도 됨

```
int num= 5;

for(;  num<3;num++)
```

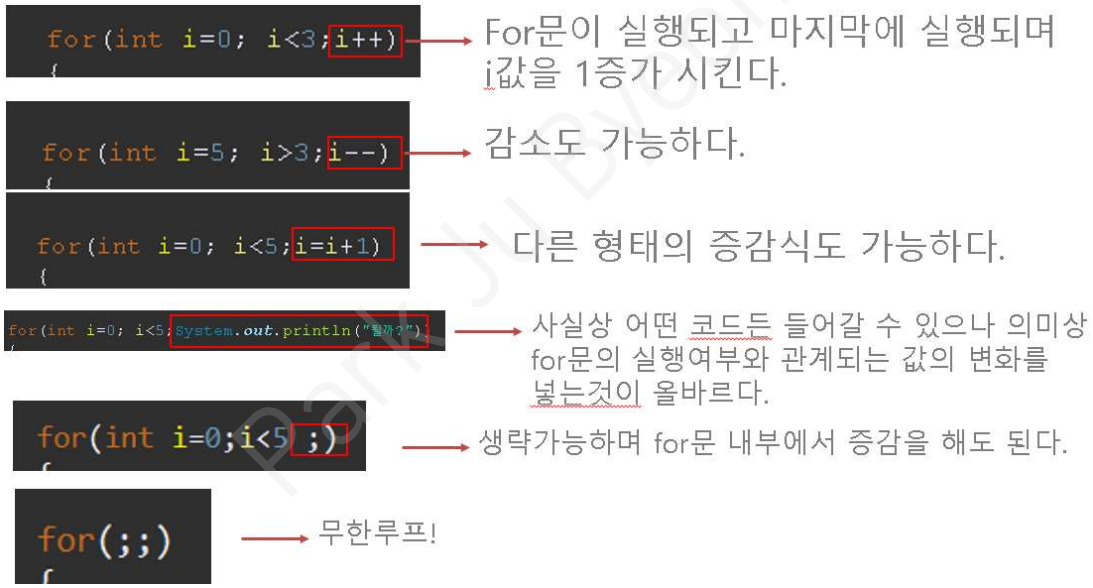
조건식 과 증감식이 외부에서 선언된 변수를 활용한다면 없어도 된다.

2.2 조건식

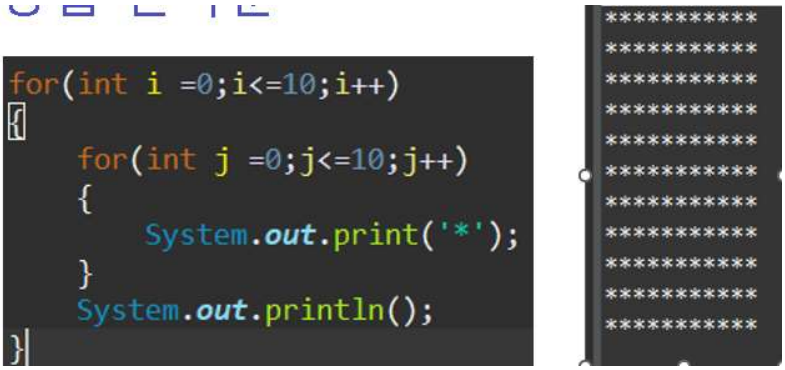
true, false의 결과가 나오는 조건식이 들어가야만 한다. 해당 조건식이 true 이면 for문의 영역 부분이 실행된다.

false라면 for문을 종료하고 중괄호가 닫히는 다음부터 실행순서가 넘어간다.
아래는 조건식 부분의 여러 가지 활용 방법들이다.
*무한루프: 조건식이 false가 될 경우가 없어 무한히 반복문이 돌게 되는 것을 말한다. 의도치 않은 무한루프는 강제종료를 해야 한다.

2.3 증감식
for문의 조건식이 처음엔 true였다가 일정 횟수이후 false가 나오게 하려면 반복문이 실행될 때마다 값의 변화를 주어야 한다. 이러한 역할을 할 코드를 넣는곳 이다.



2.4 중첩 반복문
반복문 역시 이전에 배웠든 if, switch 문과 같이 내부에 중첩으로 사용할 수 있다.
아래의 그림은 for문을 중첩으로 하여 별로 사각형을 출력하는 코드이다. 실행순서는 for문과 똑같으며 내부에 있는 for문이 10번 수행되고 끝나야 외부 for문이 한번 수행된다는 것을 눈여겨 보자



3.while문

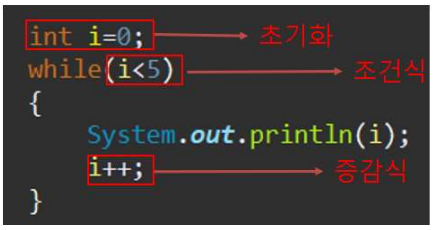
for문과 동일하게 반복적으로 코드를 수행하는 역할을 한다.

while(조건식)

{

}

위와 같은 형태로 쓰이며 if문과 다르게 문법적으론 조건식만 있으면 된다. 아래의 그림과 같이 for문과 똑같이 동작하게 만들수도 있다.while문은 for문과 다르게 조건식 생략이 불가능하며 true, false 등을 리터럴로 직접 적어줄 수 있다.




```
for(int i=0;i<5;i++)
{
    System.out.println(i);
}
```

여기서 한번 생각해 봐야 할 것은 for문이 있는데 왜 while문이 필요할까 이다. 그 질문에 대한 답은 for문은 초기화, 조건식, 증감식이 정형화된 틀이라는 것이다. 이러한 틀은 대게 정해진 횟수만큼 수행되어야 할 때 유용할 것이다. 반면에 while문의 경우 특정 상태일때만 수행한다 라는 조건에 주로 쓰인다.

아래의 예시처럼 게임 케릭터의 체력이 있을때만 수행한다 라고 한다면 몇 번 반복되어야 하는지 명확하지가 않다. 이럴 때 주로 while문을 사용한다.

for

횟수가 정해져 있을때 사용한다

```
for(int i = 1 ;i<=5; i++)
{
    System.out.println(i+" 번째 패스워드를 틀렸습니다."+ (5-i)+"번 남았습니다.");
}
```

while

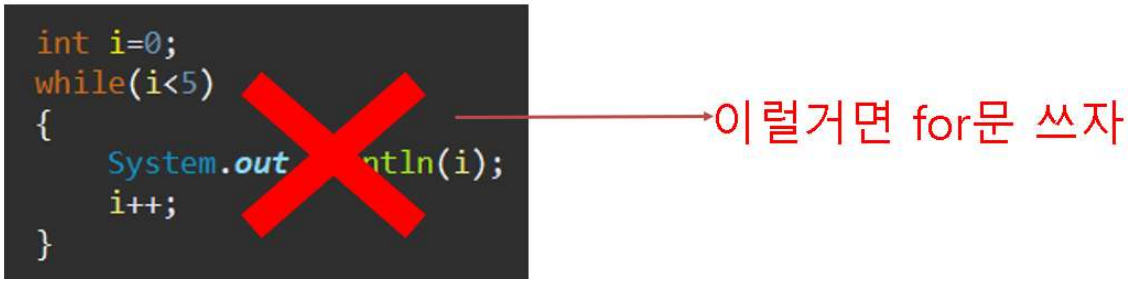
특정 조건을 만족할때까지일때 사용한다

```
int life = 100;


while(life >0) // 케릭터의 체력이 있을때만
{

}
```

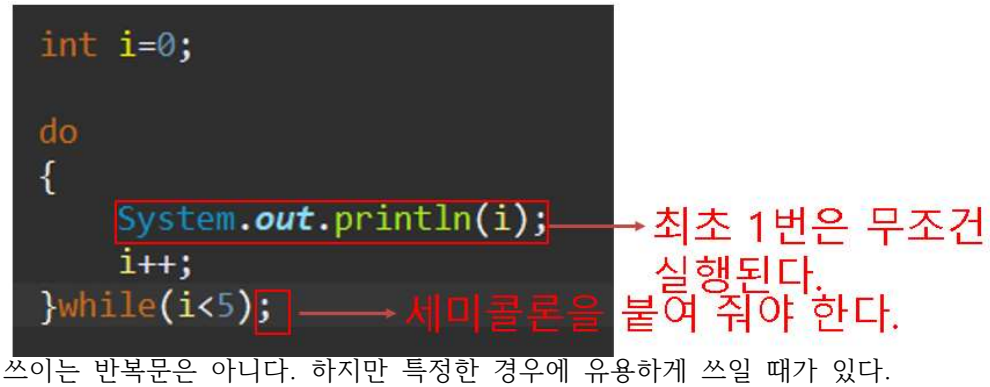

따라서 사실상 아래와 같은 예시의 경우에는 while보다는 그냥 for문을 사용하는 것이 바람직하다.



4.do ~ while문

while문과 거의 유사하며 앞에 do 영역이 생기기만 할뿐이다. 차이점은 while의 경우 조건문을 먼저 따져보고 false 가 된다면 한번도 실행하지 않고 넘어 간다.

반면 do~while의 경우 do 영역이 먼저 나오고 조건식이 가장 뒤에 있다. 따라서 do 영역을 일단 한번 실행하고 난뒤 조건식을 판단한다 이때 false 가 나오면 그대로 종료되고 true면은 다시 do영역을 실행한다.



사실 실무에서 많이 쓰이는 반복문은 아니다. 하지만 특정한 경우에 유용하게 쓰일 때가 있다.

5.break VS continue

break문은 switch를 배울 때 봤을 것이다. 해당 문장을 만나면 실행이 종료되고 해당영역 밖으로 벗어난다.

반복문에서도 마찬가지이다. 반복문 내에서 break를 만나면 즉시 반복문을 종료한다.

아래의 그림처럼 while문 밖으로 벗어나는 것이다.

하지만 continue의 경우 반복문 내의 끝지점으로 이동한다. 즉 아래의 for문의 경우 continue;를 만나면 그뒤의 문장들은 모두 지나가고 반복문의 끝으로 이동한다. 그리고 i++를 실행하여 I의값을 1 증가 시키고 다시 반복문을 이어 나간다.