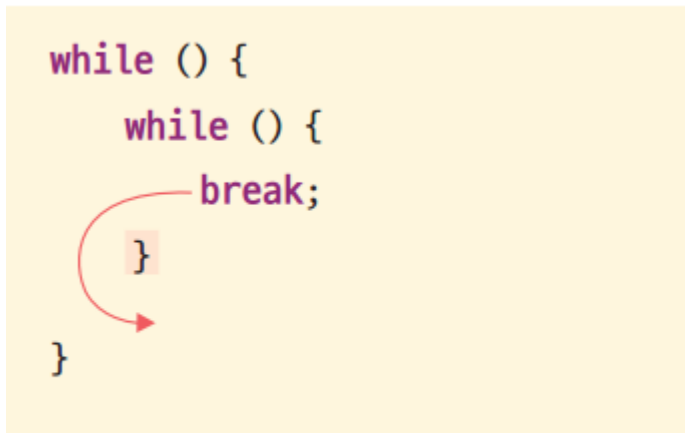


제어문과 메서드

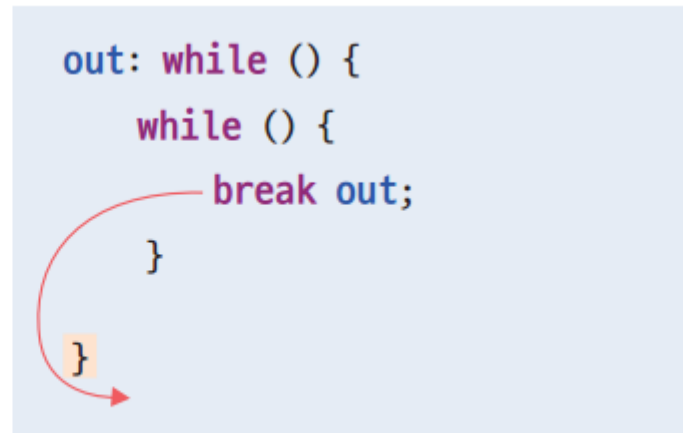


분기문

■ break 문

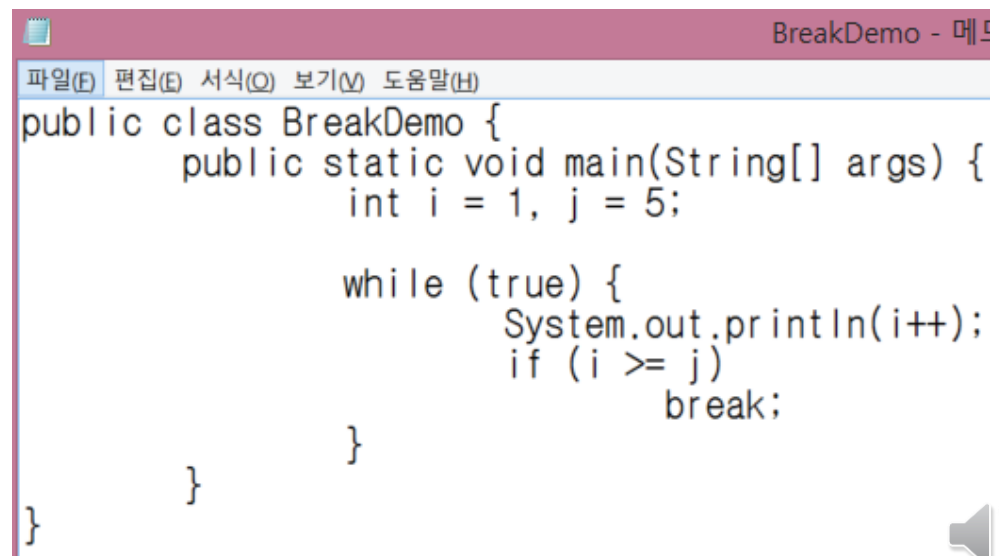


(a) break를 포함한 맨 안쪽 반복문 종료



(b) 레이블이 표시된 반복문 종료

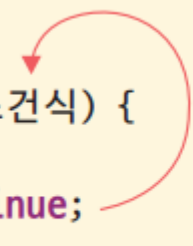
- 예제 : [sec04/BreakDemo](#)



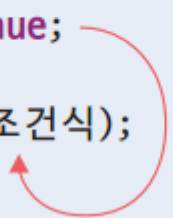
분기문

■ continue 문


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



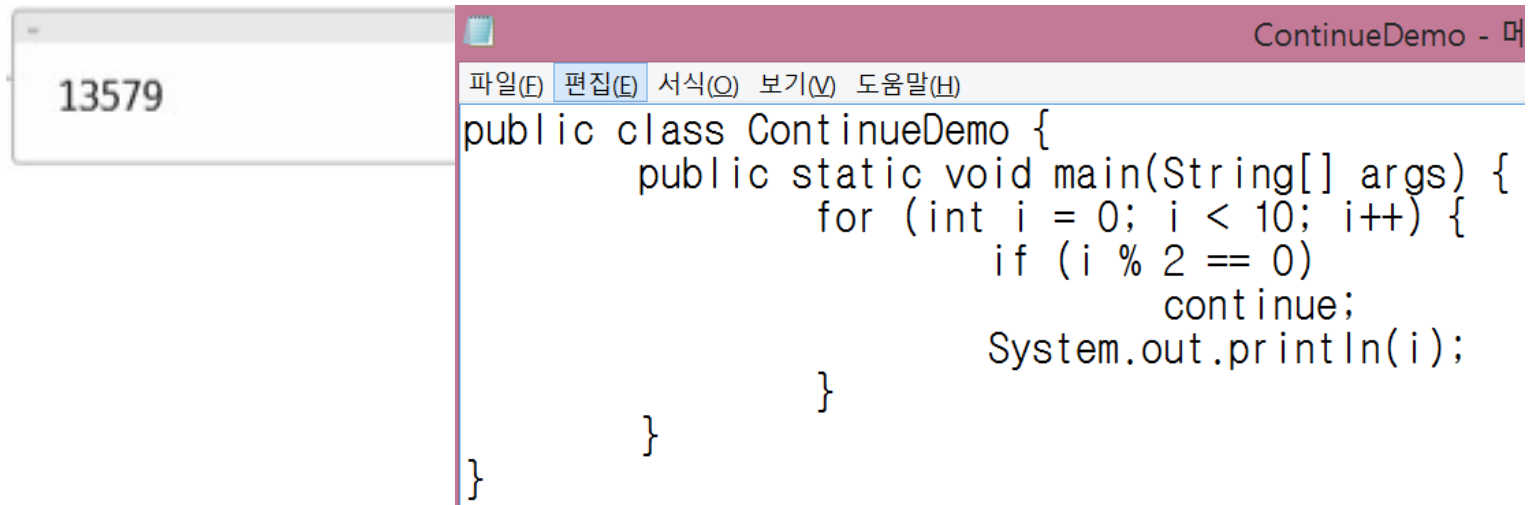
```
do {  
    continue;  
} while (조건식);
```



```
for (초기식; 조건식; 증감식) {  
    continue;  
}
```



- 예제 : [sec04/ContinueDemo](#)



```
ContinueDemo - 머  
파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)  
public class ContinueDemo {  
    public static void main(String[] args) {  
        for (int i = 0; i < 10; i++) {  
            if (i % 2 == 0)  
                continue;  
            System.out.println(i);  
        }  
    }  
}
```



메서드

■ 필요성

- 메서드를 이용하지 않은 예제 : [sec06/Method1Demo](#)
- 메서드를 이용한 예제 : [sec06/Method2Demo](#)

```
합(1~10) : 55
합(10~100) : 5005
합(100~1000) : 495550
```

```
Method1Demo - 메모장
파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)
public class Method1Demo {
    public static void main(String[] args) {
        int sum = 0;
        for (int i = 0; i <= 10; i++)
            sum += i;
        System.out.println("합(1~10) : " + sum);

        sum = 0;
        for (int i = 10; i <= 100; i++)
            sum += i;
        System.out.println("합(10~100) : " + sum);

        sum = 0;
        for (int i = 100; i <= 1000; i++)
            sum += i;
        System.out.println("합(100~1000) : " + sum);
    }
}
```

```
Method2Demo - 메모장
파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)
public class Method2Demo {
    public static void main(String[] args) {
        System.out.println("합(1~10) : " + sum(1, 10));
        System.out.println("합(10~100) : " + sum(10, 100));
        System.out.println("합(100~1000) : " + sum(100, 1000));
    }

    public static int sum(int i1, int i2) {
        int sum = 0;
        for (int i = i1; i <= i2; i++)
            sum += i;

        return sum;
    }
}
```

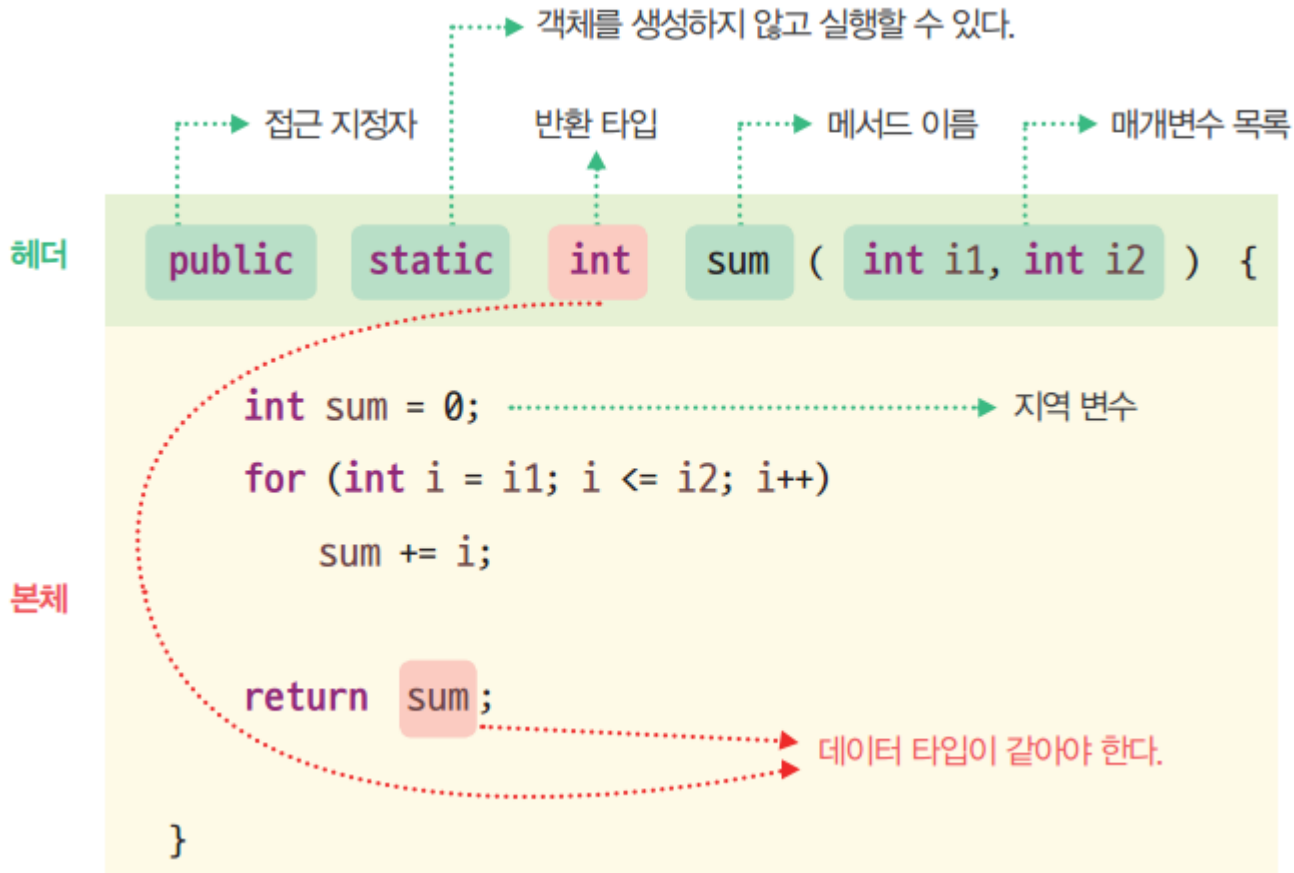
■ 메서드를 이용하면 얻을 수 있는 장점

- 중복 코드를 줄이고 코드를 재사용할 수 있다.
- 코드를 모듈화해 가독성을 높이므로 프로그램의 품질을 향상시킨다.



메서드

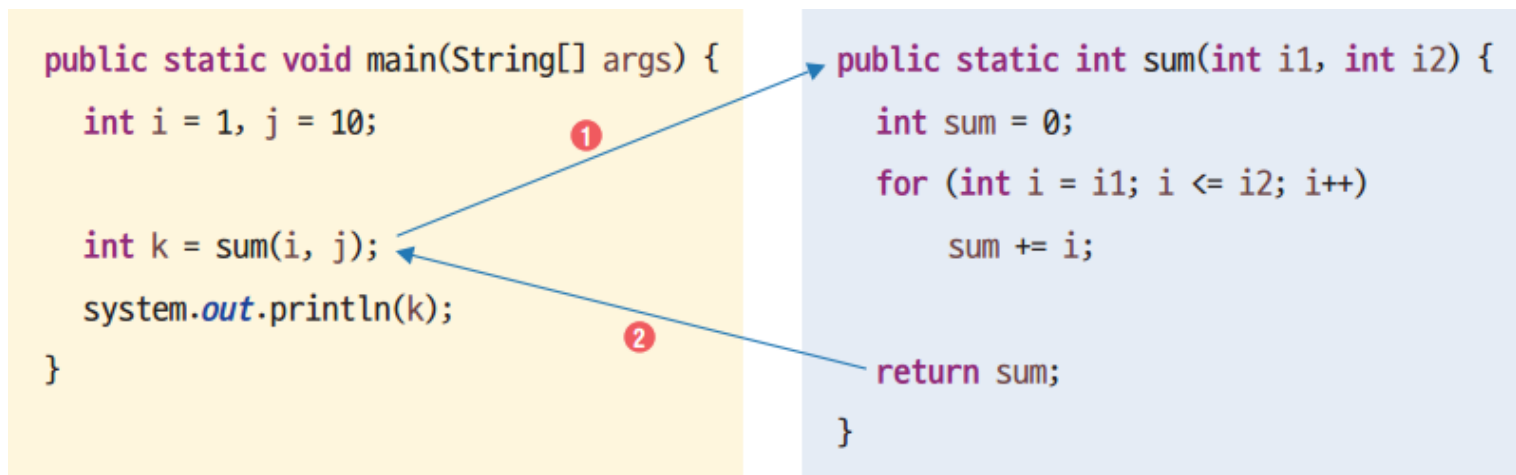
■ 메서드의 구조



메서드

■ 메서드의 호출과 반환

- 메서드를 호출하면 제어가 호출된 메서드(callee)로 넘어갔다가 호출된 메서드의 실행을 마친 후 호출한 메서드(caller)로 다시 돌아온다. 단, return 문을 사용하면 다음과 같이 메서드의 실행 도중에도 호출한 메서드로 제어를 넘길 수 있다.



- 예제 : [sec06/ReturnDemo](#)



```
package sec06;  
  
public class ReturnDemo {  
    public static void main(String[] args) {  
        printScore(99);  
        printScore(120);  
    }  
  
    public static void printScore(int score) {  
        if (score < 0 || score > 100) {  
            System.out.println("잘못된 점수 : " + score);  
            return;  
        }  
        System.out.println("점수 : " + score);  
    }  
}
```



메서드

■ 메서드의 매개변수

- 예제 : [sec06/EchoDemo](#)



```
EchoDemo - 메모장
파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)
public class EchoDemo {
    public static void main(String[] args) {
        echo("안녕!", 3);
    }

    public static void echo(String s, int n) {
        for (int i = 0; i < n; i++)
            System.out.println(s);
    }
}
```



메서드

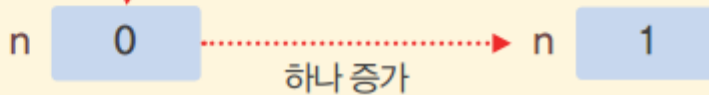
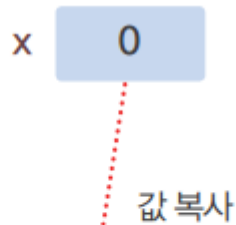
■ 값 전달(call by value)

- 예제 : [sec06/IncrementDemo](#)

```
int x = 0;  
increment(x);  
// x는 여전히 0
```

```
increment(int n)
```

```
n++;
```



increment() 메서드를 호출하기 전의 x는 0
increment() 메서드를 시작할 때의 n은 0
increment() 메서드가 끝날 때의 n은 1
increment() 메서드를 호출한 후의 x는 0

```
IncrementDemo - 메모장  
파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)  
public class IncrementDemo {  
    public static void main(String[] args) {  
        int x = 0;  
        System.out.println("increment() 메서드를 호출하기 전의 x는 " + x);  
        increment(x);  
        System.out.println("increment() 메서드를 호출한 후의 x는 " + x);  
    }  
  
    public static void increment(int n) {  
        System.out.println("increment() 메서드를 시작할 때의 n은 " + n);  
        n++;  
        System.out.println("increment() 메서드가 끝날 때의 n은 " + n);  
    }  
}
```



메서드

■ 메서드 오버로딩

- 메서드 시그너처(Method Signature) : 메서드 이름, 매개변수의 개수, 매개변수의 타입과 순서를 의미
- 메서드 이름은 같지만 메서드 시그니처가 다른 메서드를 정의하는 것을 메서드 오버로딩(Method Overloading)이라고 한다.

- 예제 : [sec06/OverloadDemo](#)

```
max(3, 7) = 7
max(7.0, 3.0) = 7.0
max(3, 7, 10) = 10
```

```
OverloadDemo - 메모장
파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

public class OverloadDemo {
    public static void main(String[] args) {
        int i1 = 3, i2 = 7, i3 = 10;
        double d1 = 7.0, d2 = 3.0;

        System.out.printf("max(%d, %d) = %dWn", i1, i2, max(i1, i2));
        System.out.printf("max(%.1f, %.1f) = %.1fWn", d1, d2, max(d1, d2));
        System.out.printf("max(%d, %d, %d) = %dWn", i1, i2, i3, max(i1, i2, i3));
    }

    public static int max(int n1, int n2) {
        int result = n1 > n2 ? n1 : n2;
        return result;
    }

    public static double max(double n1, double n2) {
        double result = n1 > n2 ? n1 : n2;
        return result;
    }

    public static int max(int n1, int n2, int n3) {
        return max(max(n1, n2), n3);
    }
}
```



Switch 문

■ 기초

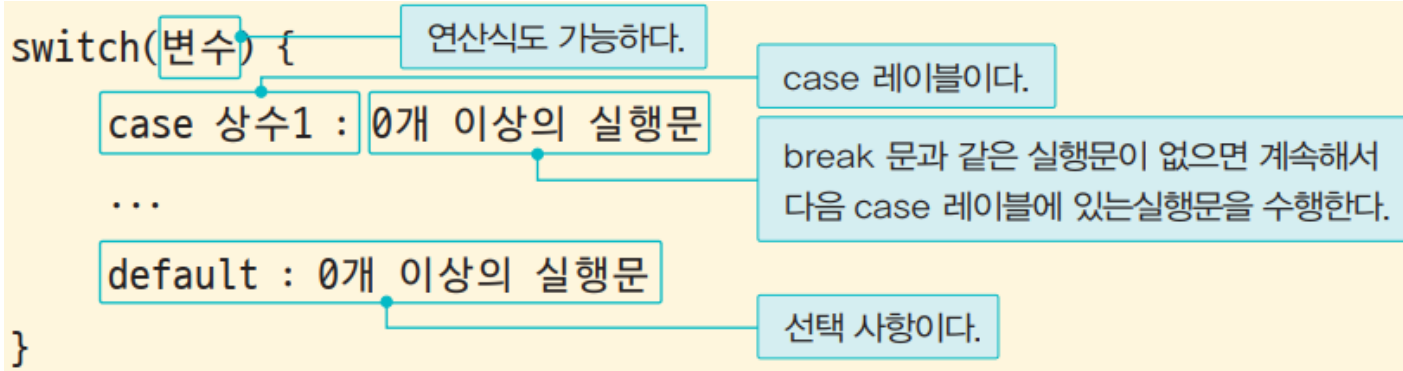


- switch 문은 if 문과 마찬가지로 조건문의 일종
- 여러 경로 중 하나를 선택할 때 사용
- 기존 switch 문은 낙하 방식으로 콜론 case 레이블 이용
- 자바 14부터는 비낙하 방식의 화살표 case 레이블 도입, switch 연산식 가능



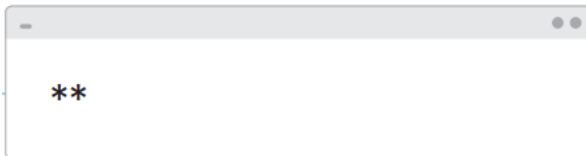
Switch 문

■ 콜론 레이블을 사용하는 기존 switch 문



- 0개 이상의 case 절과 0이나 1개의 default 절로 구성
- Switch 변수로 정수 타입만 사용할 수 있었지만, 자바 7부터는 문자열과 열거 타입도 사용 가능

● 예제 : [sec05/Switch1Demo](#)



```
Switch1Demo - 메모장
파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)
public class Switch1Demo {
    public static void main(String[] args) {
        int number = 2;

        switch (number) {
            case 3:
                System.out.print("*");
            case 2:
                System.out.print("*");
            case 1:
                System.out.print("*");
        }
    }
}
```



Switch 문

■ 개선된 switch 문

- 필요성 : 깔끔하지 못하고 가독성도 떨어지며, break문의 누락으로 인한 오류 가능성도 크다
- 자바 14부터 다음과 같은 변화를 도입

- 화살표 case 레이블
- Switch 연산식
- 다중 case 레이블
- Yield 예약어

호랑이는 포유류이다.

참새는 조류이다.

고등어는 어류이다.

어이쿠! 곰팡이는 ...이다.

- 예제 : [sec05/Switch3Demo](#)(switch 문), [sec02/Switch4Demo](#)(switch 연산식)

```
Switch3Demo - 메모장
파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)
public class Switch3Demo {
    public static void main(String[] args) {
        wholslt("호랑이");
        wholslt("참새");
        wholslt("고등어");
        wholslt("곰팡이");
    }

    static void wholslt(String bio) {
        String kind = "...";
        switch (bio) {
            case "호랑이", "사자" -> kind = "포유류";
            case "독수리", "참새" -> kind = "조류";
            case "고등어", "연어" -> kind = "어류";
            default -> System.out.print("어이쿠! ");
        }
        System.out.printf("%s는 %s이다.\n", bio, kind);
    }
}
```

```
Switch4Demo - 메모장
파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)
public class Switch4Demo {
    public static void main(String[] args) {
        wholslt("호랑이");
        wholslt("참새");
        wholslt("고등어");
        wholslt("곰팡이");
    }

    static void wholslt(String bio) {
        String kind = switch (bio) {
            case "호랑이", "사자" -> "포유류";
            case "독수리", "참새" -> "조류";
            case "고등어", "연어" -> "어류";
            default -> {
                System.out.print("어이쿠! ");
                yield "...";
            }
        };
        System.out.printf("%s는 %s이다.\n", bio, kind);
    }
}
```



Switch 문

■ 개선된 switch 문

- 자바 14부터는 기존 switch 문도 연산식, 다중 case 레이블, yield 예약어를 허용

```
String kind = switch (bio) {  
    case "호랑이", "사자":  
        yield "포유류";  
    case "독수리", "참새":  
        yield "조류";  
    case "고등어", "연어":  
        yield "어류";  
    default:  
        System.out.print("어이쿠! ");  
        yield "...";  
};
```

기존 switch 문에서는 블록이 아니더라도 yield 예약어를 사용할 수 있다.



Switch 문

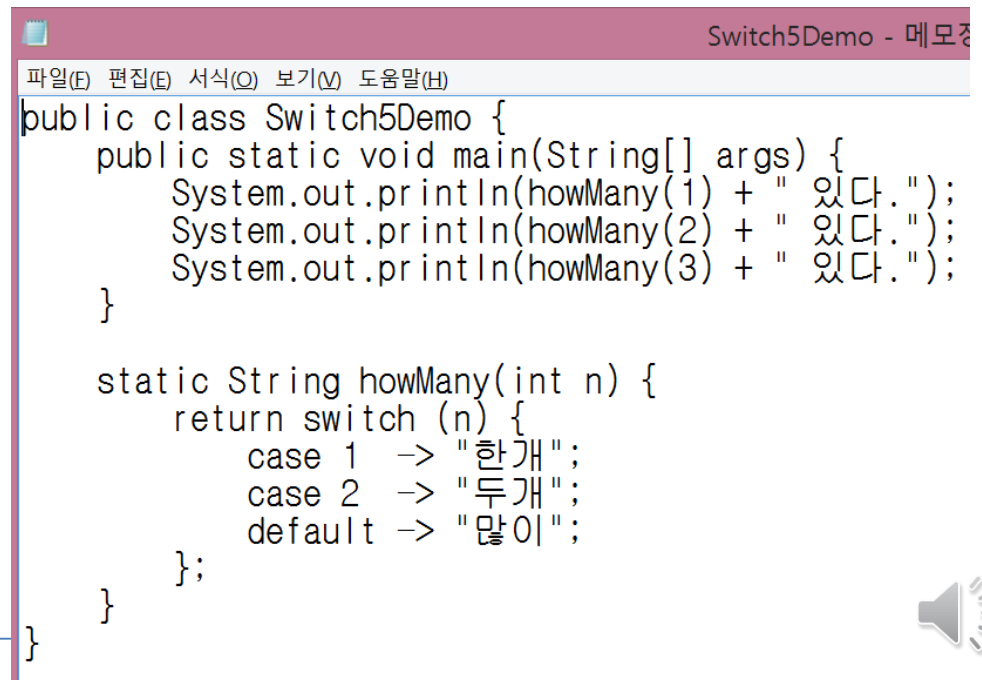
■ Switch 연산식의 주의 사항

- 가능한 모든 값에 대하여 일치하는 case 레이블이 없으면 오류가 발생
- 다음 코드에서 변수 n의 모든 가능한 값은 정수이므로 오류 발생

```
static String howMany(int n){  
    return switch(n){  
        case 1 -> "1개";  
        case 2 -> "2개";  
    }; // default 문은 선택 사항  
}
```

- 예제 : [sec05/Switch5Demo](#)

```
1개 있다.  
2개 있다.  
많이 있다.
```



```
Switch5Demo - 메모장  
파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)  
public class Switch5Demo {  
    public static void main(String[] args) {  
        System.out.println(howMany(1) + " 있다.");  
        System.out.println(howMany(2) + " 있다.");  
        System.out.println(howMany(3) + " 있다.");  
    }  
  
    static String howMany(int n) {  
        return switch (n) {  
            case 1 -> "한개";  
            case 2 -> "두개";  
            default -> "많이";  
        };  
    }  
}
```

