◆ **제어문(flow control statement)**: 조건문과 반복문이 있으며, 프로그램의 흐름(flow)을 바꾸는 역할을 한다. 조건문은 조건에 따라 다른 문장이 수행되고, 반복문은 특정 문장들을 반복해서 수행한다.

● 조건문: if문과 switch문이 있으며 처리할 경우의 수가 많을 때는 switch문이 효율적이지만 제약이 많다.

1. if문
- 만일(if) 조건이 참(true)이면 괄호{}안의 문장들을 수행한다.
- 블럭(block): 여러 문장을 하나의 단위로 묶는 괄호{}. 끝에 ;을 붙이지 않는다. 블럭 내의 문장들은 탭(tab)으로 들여쓰기(indentation)을 해서 블럭 안에 속한 문장이라는 것을 표시해 준다. 만일 블럭 내의 문장이 하나뿐일 때는 괄호{}를 생략할 수 있다.

▶ if-else문
- 조건식의 결과가 거짓ㅇ리 때 else블럭의 문장을 수행한다
- if문을 두 개 사용할 경우 두 개의 조건식을 계산해야 하지만, if-else문을 사용할 경우 하나의 조건식만 계산하면 되므로 더 효율적이고 간단하다.
- 블럭 내의 문장이 하나일 경우 괄호{}를 생략할 수 있다.

▶ if-else if문
- 여러 개의 조건식을 쓸 때 사용한다.
- 첫 번째 조건식부터 순서대로 평가해서 결과가 참인 조건식을 만나면, 해당 블럭{}만 수행하고 "if-else if"문 전체를 벗어난다.
- 만일 결과가 참인 조건식이 하나도 없으면, 마지막에 있는 else블럭의 문장들이 수행된다. 그리고 else블럭은 생략이 가능하다.

▶ 중첩 if문
- if문의 블럭 내에 또 다른 if문을 포함시키는 것.

```
if (조건식1) {
    // 조건식1의 연산결과가 true일 때 수행될 문장들을 적는다.
    if (조건식2) {
        // 조건식1과 조건식2가 모두 true일 때 수행될 문장들
    } else {
        // 조건식1이 true이고, 조건식2가 false일 때 수행되는 문장들
    }
} else {
    // 조건식1이 false일 때 수행되는 문장들
}
```

2. switch 문
- 조건식의 결과가 참과 거짓, 두 가지 밖에 없기 때문에 경우의 수가 많아질 수록 else-if를 계속 추가해야 해서 조건식이 많아지므로 복잡해지는 if문과는 달리, swtich문은 단 하나의 조건식으로 많은 경우의 수를 처리할 수 있고, 표현도 간결하므로 알아보기 쉽다. 그래서 처리할 경우의 수가 많은 경우에는 if문보다 switch문으로 작성하는 것이 좋다.
- 순서: 1) 조건식을 계산한다. 2) 조건식의 결과와 일치하는 case문으로 이동한다. 3) 이후의 문장들을 수행한다. 4) break문이나 switch문의 끝을 만나면 switch의 전체를 빠져나간다.

- 맨 마지막의 default는 조건식의 결과가 일치하는 case이 하나도 없을 경우에 이동한다. 보통 마지막에 위치하기 때문에 break문을 쓰지 않아도 된다.
- break문은 각 case문의 영역을 구분하는 역할을 하며, 만일 생략하면 case문 사이의 구분이 없어지므로 다든 break문을 만나거나 swtich문 블럭{}의 끝을 만날 때까지 나오는 모든 문장들을 수행한다.

```
// 다음과 같이 고의적으로 break문을 생략하는 경우도 있다.
switch (level) {
    case 3:
        grantDelete();        // 삭제 권한을 준다.
    case 2:
        grantWrite();        // 쓰기 권한을 준다.
    case 1:
        grantRead();        // 읽기 권한을 준다.
}
// 로그인한 사용자의 등급을 체크하여 제일 높은 등급인 3을 가진 사용자는 삭제, 쓰기, 읽기 권한
// 모두를 갖게 되고, 제일 낮은 등급인 1을 가진 사용자는 읽기 권한만을 갖게 된다.
```

- 조건: 1) swtich문의 조건식의 결과값은 반드시 정수 또는 문자열이어야 한다. 2) case문의 값 역시 정수, 정수 상수 (대문자 변수), 문자열, 또는 문자 리터럴(사실은 정수인 유니코드로 저장되기 때문)이어야 하고, 중복되지 않아야 한다.
- case문은 한 줄에 하나씩 쓰던, 한 줄에 붙여서 쓰던 관계 없다.

```
switch (result) {
    case 1:
    case 2:
    case 3:
        System.out.println("A+ 입니다");
        break;
}
```

```
switch (result) {
    case 1: case 2: case 3:
        System.out.println("A+ 입니다");
        break;
}
```

- switch문은 조건식을 1번만 계산하면 되므로 더 빠르다. 반드시 속도를 향상시켜야 한다면 복잡하더라도 switch를 선택할 수 있지만, 그렇지 않다면 if문이 더 적합한 경우도 있다. switch문에서는 조건식을 잘 만들어서 case의 갯수를 줄이는 것이 중요하다.
- swtich문의 중첩: 중첩이 가능하다. 한 가지 중요한 점은 중첩 switch문의 블럭 뒤에 반드시 break문을 빼먹지 않아야 한다는 점이다.

• • • • •

```
// 임의의 정수(난수) 구하기
// Math.random(): 0.0~1 사이의 범위에 속하는 하나의 double 값을 반환한다.
// 이때 0은 포함 되고, 1은 포함되지 않는다 (0.0~0.9999~)
// 1과 3 사이의 정수 구하기를 원한다면, 다음과 같은 과정을 거치면 얻을 수 있다.
// Math.random() * 3을 한다. 이때 범위는 다음과 같다: 0.0~3.0
// 이 수를 int로 캐스팅한다. (int) (Math.random() * 3). 이때 범위는: 0부터 3까지다.
// 각 변에 1을 더한다. (int) (Math.random() * 3) + 1. 이때 범위는: 1부터 4까지다.
// 1은 포함되고, 4는 포함되지 않는다. 따라서 1과 3 사이의 정수 중 하나를 얻을 수 있다.
// 같은 방식으로 주사위를 던졌을 때 나타나는 임의의 값을 얻기 위해서는 3대신 6을 곱하면 된다.
// 만약 -3부터 0까지의 범위의 정수 중 하나를 얻으려면,
// + 1 대신 - 3을 해 주면 된다. (int) (Math.random() * 3) + -3
```

```
// 문자열에 저장된 문자는 '문자열.charAt(index)'로 가져올 수 있다.
// 이때 index는 0부터 시작한다.
```

Resource: 자바의 정석