다중 선택이 가능한 switch ~ case 조건문

형식) **switch(정수식) {**
case 값2 : 처리할 문장 2; break;
...중략
default: 처리할 문장;
}

정수식과 case문의 값과 일치하면 **해당 처리할 문장이 실행**된다. 그리고 문장 실행 후 **break문을 만나면 switch~case문을 종료** 한다.

그리고 **default문은 해당 조건이 없을때 실행된다.**

참고) jdk1.7 이후부터는 **switch 문의 조건식**에도 **문자열**이 올수 있다. **case문의 값도 문자열이 올 수 있다.**

새로운 switch 표현식 (Java 12)

Java 12에서 도입된 switch 표현식은 switch 문을 더 간결하게 만들 수 있는 기능이다. Java 12부터는 switch 문을 표현식(expression)으로 사용할 수 있다. 이를 통해 switch 문에서 값을 반환하거나 변수를 직접 초기화할 수 있다.

```
package 람다식;

public class NewSwitch {
	public static void main(String[] args) {

		int day = 2;
		String dayName = switch (day) {
		    case 1 -> "Monday";
		    case 2 -> "Tuesday";
		    case 3 -> "Wednesday";
		    default -> "Unknown";
		};
		System.out.println(dayName);
	}
}
```

특징

화살표 연산자 (->) 사용:

case 레이블 뒤에 화살표 연산자(->)를 사용하여 반환 값을 지정한다.

이는 switch 블록을 더 간결하고 명확하게 만들어 준다.

문장 또는 표현식으로 결과 반환:

switch 표현식은 결과를 반환할 수 있다. 위의 예에서는 dayName 변수를 초기화하는 데 사용되었다.

switch 문 자체가 값을 반환하는 표현식으로 사용된다. break문을 없애는 대신에 화살표와 중괄호를 사용한다.

블록을 사용할 수도 있다.
복잡한 로직이 필요한 경우, 블록 {}을 사용하여 여러 문장을 포함할 수 있다.
예를 들어:

```
System.out.println("\n====================>\n");

                var day2=1;
                String dayName2 = switch (day2) {
                case 1 -> {
                        String name = "Monday";
                        yield name; // yield 키워드를 사용하여 값을 반환
                }
                case 2 -> "Tuesday";
                case 3 -> "Wednesday";
                default -> "Unknown";
                };

                System.out.println("반환된 요일은 "+dayName2);
```

yield 키워드(Java 13에서 도입):
switch 블록 내에서 값을 반환하려면 yield 키워드를 사용하여 결과를 반환한다. 이 기능은 블록에서 반환 값을 명시적으로 지정할 수 있게 해 준다. 단, 이 경우에는 default문이 반드시 있어야 한다.

자바 17까지는 표현값이 null일 경우 switch 문에서 NullPointerException 예외오류가 발생했지만, 자바 21부터는 null을 지정해도 예외 오류가 발생하지 않고 처리할 수 있게 되었다.

```
package 새로운문법;

public class Java21SwitchExample {
        private static void method01(String s) {
                switch(s) {
                 case null -> System.out.println("null");
                 case "a", "b" -> System.out.println("a or b");
                 case "c" -> System.out.println("c");
                 default -> System.out.println("해당 사항 없다.");
                }
}
```

```
        private static void method02(String s) {
                switch(s) {
                 case "a" , "b" -> System.out.println("a or b");
                 case "c" -> System.out.println("c");
                 case null, default -> System.out.println(" null or 해당 사항 없다.");
                }
        }

        public static void main(String[] args) {
           method01(null);
           method02("d");
           method02(null);
        }
}
```

자바 21에서 새롭게 추가된 패턴 매칭

자바 21부터는 switch 레이블에 패턴과 가드를 작성해서 표현값과 매칭시킬 수도 있다. 이 방식은 표현값이 객체를 참조하는 변수일 경우에만 사용할 수 있다.

```
강화된 방식1(자바 21부터)
switch(표현값){
  case 패턴 [가드] -> {
실행문; ...
  }
...
  case null [, default] -> {
    실행문; ..
  }
  default -> {
   실행문; ...
 }
}
```

```
강화된 방식2(자바 21부터)
 타입 변수 = switch(표현값){
     case 패턴 [가드] -> 리턴값;
 case 패턴 [가드] -> {
    ..;
    yield 리턴값;
    }
....
 case null[, default] -> 리턴값;
   default -> 리턴값
};
```

표현값이 참조 타입 변수일 경우 패턴을 사용해서 타입 검사를 수행하고, 자동 타입 변환해서 패턴 변수를 초기화 시킨다. 그리고 패턴 변수를 중괄호 {} 블록에서 사용할 수 있다.

```
switch(object){
  case Integer i -> {  //i변수 사용 }  //object이 Integer 타입인 경우 실행(자동 타입
//변환
  case String s -> { //s변수 사용 } //object이 String 타입인 경우 실행(자동 타입 변
//환
```

표현값과 패턴 중 하나와 반드시 매칭되도록 해야 한다. 만약 매칭할 패턴이 없는 경우에는 나머지 매칭을 위해 default문을 포함해야 한다. 이것을 표현값과 실행문의 완전성이라고 하는데, 표현값이 반드시 실행문에서 처리되어야 함을 뜻한다.

```
package 새로운문법;

import java.util.Date;

public class Java21SwitchExample02 {
	public static void method01(Object obj) {
		switch(obj) {
		 case Integer i -> System.out.println(i);
		 case String s -> System.out.println("\"" + s +"\"");
		 case null, default -> System.out.println("null or 해당 사항 없다.");
		}
	}

	private static void method02(Object obj) {
		String result = switch(obj) {
		 case Integer i -> String.valueOf(i);//인자값을 문자열로 변환
		 case String s -> "\"" + s + "\"";
		 case null, default -> "unknown";
		};
		System.out.println(result);
	}
	public static void main(String[] args) {
	method01(100);
	method01("100");
	method01(null);
	method01(new Date());

	System.out.println("\n");

	method02(10);
	method02("10"); method02(null); method02(new Date());
	}
}
```

자바 21에서 가드 사용

패턴과 함께 좀 더 상세한 일치 조건을 만들기 위해서 when으로 시작하는 가드(guard)를 사용할 수 있다. when 다음에는 패턴 변수를 사용해서 boolean 타입을 리턴하는 조건식 또는 메서드 호출 코드가 올 수 있다. true를 리턴하면 레이블을 선택되고 중괄호 실행문장이 수행된다.

```
switch(object){
 case Integer a when a > 0 -> { //object이 Integer 타입이면서 양수일 때 선택
  ...
 }
 case String s when s.equals("a") -> { //object이 String 타입이면서 "a"일 때 선택
 ...
}
```

```
package 새로운문법;

public class Java21SwitchExample03 {
    private static void method01(Object obj) {
        int score = switch(obj)  {
                case Integer i when i == 1 -> 90;
                case Integer i when i == 2 -> 80;
                case Integer i -> 70;
                case String s when s.equals("a") -> 90;
                case String s when s.equals("b") -> 80;
                case String s -> 70;
                case null, default -> 0;
        };
        System.out.println("score = "+score);
    }
        public static void main(String[] args) {
        method01(1); method01(2); method01(3);
        method01("a"); method01("b"); method01("c");
        method01(null);
        }
}
```

레이블 통과

레이블에 패턴이 사용되면 기본적으로 다음 레이블로 통과가 금지된다. 단, 다음이 default 문이라면 통과할 수 있지만 화살표가 사용되면 무조건 통과가 금지된다.

```
package 새로운문법;

public class Java21SwitchExample04 {
```

```
    private static void method01(Object obj) {
        switch(obj) {
         case String s:
                System.out.println("String : " + s);
                break;//생략하면 컴파일 에러가 발생(다음 case 레이블로 통과 금지)
         case Integer i:
                System.out.println("Ineger:"+i);//단, 다음이 default 문이라면 통과
가능
         case null,default:
                System.out.println("null or unknown");
        }
    }

    private static void method02(Object obj) {
        switch(obj) {
         case String s -> System.out.println("String : "+s);
         case Integer i -> System.out.println("Integer i="+i);//레이블에 패턴이 사용
되면 화살표 연산자(->)에서는 통과금지
         case null,default -> System.out.println("null or unknown");
        }
    }
        public static void main(String[] args) {
        method01("a");
        System.out.println();
        method01(1);
        System.out.println();
        method02(100);
        }
}
```

레이블 작성순서

레이블이 패턴일 경우에는 좁은 범위의 패턴을 먼저 작성하고, 넓은 범위의 패턴을 나중에 작성해야 한다. switch 문은 위에서부터 순차적으로 표현값과 패턴을 매핑하기 때문에 위쪽 패턴이 먼저 매칭되면 아래쪽 패턴은 검사하지 않는다.

부모와 자식 관계, 인터페이스와 구현 관계에도 범위가 있다. 부모는 자식 클래스보다 항상 넓은 범위를 가지며, 인터페이스는 구현 클래스 보다 항상 넓은 범위를 가진다. 그러므로 부모 패턴보다 자손 패턴을 먼저 작성해야 한다.

인터페이스와 그 구현 클래스 간에도 동일한 규칙이 적용된다. 구현 클래스의 패턴이 먼저 작성되어야 하며, 그 후에 인터페이스 패턴이 나와야 한다.

```
package 새로운문법;

public class Java21SwitchExample05 {
        private static void method01(Integer obj) {
                switch(obj) {
                 case 0 -> System.out.println(0);//레이블에 패턴을 사용하면 좁은 범위의 패턴을 먼저 기술하고, 넓은 범위의 패턴은 나중에 기술한다.
                  case Integer i when i > 0 -> System.out.println("positive number");//양수
                  case Integer i -> System.out.println("negative number"); //음수
                }
        }

        private static class A{}

        private static class B extends A{}

        private static void method02(Object obj) {
                 switch(obj) {
                  case B b -> System.out.println("obj is B type");//레이블에 패턴이 사용되면 부모가 자손보다 항상 넓은 범위를 가진다. 그러므로 자손을 먼저 기술한다.
                   case A a -> System.out.println("obj is A type");
                     case null, default -> System.out.println("obj is null or unkonow type");
                 }
        }
         public static void main(String[] args) {
        method01(5);
        method01(-5);
        method02(new A());
         }
}
```

for 반복문

형식)

```
for(초기치;조건식;증감식){
  조건식이 참일 동안만 반복;
}
```

가) **처음엔 초기값을 실행한다. 초기값이란 변수에 최초의 값을 저장하는 것을 말한**

다.
나)조건식을 평가하여 참일 동안 문장을 반복 실행한다.
다) 증감식을 실행한다.다시 조건식을 평가하여 참이면 문장을 반복하고, 거짓이면 for반복문을 종료 한다.

향상된 for문(확장 for문)

가) 향상된 for문은 J D K 1.5부터 배열과 컬렉션에 저장된 요소에 접근할 때 기존 for문보다 편리한 방법으로 처리할 수 있도록 새롭게 추가되었다.

나) 확장 for문 형식

```
for(타입 변수명:배열 또는 컬렉션){
 //배열 또는 컬렉션에 저장된 값이 매 반복마다 하나씩 순서대로 읽혀져
 //변수에 저장된다.
}
```

다) 확장 for문은 일반적인 for문과 달리 배열과 컬렉션에 저장된 요소들을 읽어오는 용도로만 사용할 수 있다.

while 반복문

형식)

```
while(조건식){
 조건식이 참일 동안만 반복;
 증감식;
}
```

무한 루프란?

가. 무한 루프란 조건식이 무조건 참이어서 영원히 반복하는 반복문을 뜻한다.

do~while 반복문

형식)

```
do{
조건식이 참일 동안만 반복;
증감식;
}while(조건식);
```

주의 사항) 나중에 조건식을 검사하기 때문에 조건식이 거짓이라도 무조건 한번은 반복한다는 단점이 있다. 실제 자바 프로젝트 에서는 do while반복문은 잘 사용 되지 않는다.

break, continue문

가. break문: 반복문 내에서 break문을 만나면 반복문을 중단한다. break 자신과 가장 가까운 반복문을 벗어난다.

나. continue문: 반복문안에서 이 문을 만나면 아래문장을 실행하지 않고, 다음 반복을 위해서 반복문 처음으로 돌아가서 그 다음 반복을 계속한다.

즉,continue문과 반복문 블록의 끝 '}' 사이의 문장들을 건너뛰고 반복을 이어가는 것

이다.