Professional Coding Specialist

COS Pro JAVA 1 급

3 강. 기초 문법 정리 3

1. 조건문
2. 반복문
3. API 클래스

과정 소개

COS Pro(Coding Specialist Professional) 시험은 요구사항을 분석하여 프로그램을 설계, 구현하는 능력과 주어진 프로그램을 디버깅하여 수정하는 능력을 평가하는 자격증 시험이며, COS Pro 1 급 자격증은 높은 수준의 프로그래밍 능력을 증명할 수 있다. 이번 시간에는 COS Pro JAVA 1 급 시험 대비를 위한 모의고사를 풀어보기 전에 알아야 하는 JAVA 의 기초 문법을 정리하는 세 번째 시간으로, JAVA 의 조건문, 반복문 문법을 정리하고, 자주 사용하는 API 클래스를 정리한다.

학습 목차

1. 조건문
2. 반복문
3. API 클래스

학습 목표

1. 자바의 조건문 문법을 정리한다.
2. 자바의 반복문 문법을 정리한다.
3. 자바에서 자주 사용하는 API 클래스를 정리한다.

1 조건문

1. if 문

1) 사용 형식

```
if (조건식 1)
{
    실행문 1
}
else if (조건식 2)
{
    실행문 2
}
…
else
{
    실행문 n
}
```

- 조건식으로 boolean 값을 갖는 변수 혹은 논리 연산식, 비교 연산식을 사용
- 조건식을 만족할 때 실행할 명령문이 한 줄이면 중괄호 생략 가능
- 중괄호 안의 명령문은 들여쓰기를 하는 것이 가독성을 높임

2) 사용 예

```
int score = 35;

if (score>30)
    System.out.println("우수");
else if (score>20)
    System.out.println("보통");
else
    System.out.println("노력 요함");
```

- 변수 score 의 값은 if 문의 첫 번째 조건식을 만족하기 때문에 첫 번째 if 문 아래에 있는 명령문을 실행하므로 결과로 출력되는 문자열은 “우수”

2. switch 문

1) 사용 형식

```
switch (조건값)
{
  case 값 1:
  조건값이 값 1 일 때 실행하는 명령들
  break;
  case 값 2:
  조건값이 값 2 일 때 실행하는 명령들
  break;
  …
  default:
  조건값이 어떤 case 에도 속하지 않을 때 실행하는 명령들
}
```

- 조건값이 정수, 문자, 문자열인 경우 사용 가능
- if 문보다 사용할 수 있는 범위가 적지만 if 문보다 가독성이 좋고, 컴파일러가 최적화를 쉽게 할 수 있어 실행 속도가 빠름
- 각각의 case 에는 break 명령이 마지막에 있어야 함
- 만일 각 case 에 break 가 없으면 모든 case 에 있는 명령들이 실행되는 오류 발생
- 여러 개의 case 가 같은 명령문을 사용하도록 지정할 수 있음

2) 사용 예

```
char grade = 'A';

switch(grade) {
case 'A' :
case 'B' :
case 'C' :
    System.out.println("통과");
    break;
case 'D' :
    System.out.println("조건부 통과");
    break;
default:
    System.out.println("보류");
}
```

- 현재 grade 값이 'A' 이므로 값을 'A' 로 갖는 첫 번째 case 가 실행되도록 System.out.println("통과") 를 실행

3. 삼항 연산자(ternary operator)

1) 사용 형식

(조건식) ? 반환값 1 : 반환값 2;

- 조건식의 결과가 참이면 반환값 1 을 리턴하고 그렇지 않으면 반환값 2 를 리턴
- 조건식 자리에 비교 연산식이나 논리 연산식 혹은 bool 값이 있는 변수 사용 가능
- 반환값으로 값뿐만 아니라 수식이나 메소드 호출 같은 명령문이 올 수 있음
- 조건식과 반환값을 모두 한 줄로 쓰기 때문에 조건식이나 반환값이 복잡해지면 프로그램의 가독성을 해치게 되므로 조건식이 간단하고 결과값이 단순한 경우에만 삼항 연산자를 사용하는 것을 추천

2) 사용 예

```
int num1 = 30;
int num2 = 12;

int maxNum = (num1>num2)?num1:num2;
System.out.println(num1+"과 "+num2+"중 더 큰 수="+maxNum);
```

- 조건식의 결과가 참이기 때문에 변수 maxNum 에는 num1 의 값인 30 이 저장

2 반복문

1. for 문

3) 사용 형식

```
for (초기값; 조건식; 증감식)
{
    반복할 명령문
}
```

- 인덱스 변수가 증감식에 의해 변하면서 조건식을 만족하는 동안 반복 실행
- for 문에 의해서 반복하는 횟수를 알 수 있음
- 반복 실행할 명령문이 한 줄이면 중괄호 생략 가능
- 중괄호 안의 명령문은 들여쓰기를 하는 것이 가독성을 높임
- 증감식에는 ++, --, +=, -=, *=, /= 도 사용 가능

4) 사용 예

`for(int i=1; i<10; i++) {`
`  System.out.println(i);`
`}`
	- 인덱스 변수 i 가 1 부터 시작하여 i 가 10 보다 작은 동안 1 씩 증가하면서 i 를 출력하는 것을 반복'

`for(int i=1; i<20; i+=2)` `  System.out.println(i);`	- 인덱스 변수 i 가 1 부터 시작하여 i 가 20 보다 작은 동안 2 씩 증가하며 i 를 출력하는 것을 반복

2. while 문

1) 사용 형식

```
while (조건식)
{
    반복할 명령문
    조건식의 변수 변화
}
```

- 조건식을 만족하는 동안 while 문 안의 명령문들을 반복 실행
- 대부분 조건식에서 사용하는 변수에 대한 변화식을 반복하는 명령문에 포함
- 반복문을 시작하기 전에 조건식이 참(true)인지 먼저 판단
- while 문은 조건식을 만족하면 계속 실행되기 때문에 총 반복 횟수를 모르는 경우도 있는데 반해, for 문은 인덱스 변수를 일정한 증감식을 이용해서 값을 바꾸며 반복 실행하기 때문에 총 반복 횟수를 알 수 있음

2) 사용 예

```
int i=0;
while(i<=10) {
    System.out.println(i);
    i+=2;
}
```

- 변수 i 의 값이 10 보다 작거나 같은 동안 i 를 출력하고 i 를 2 만큼 증가하는 것을 반복 실행
  ➔ 실행하면 10 보다 작은 짝수 출력
- while 문 안에 i += 2 가 없다면, i 는 계속 0 이고, 그렇게 되면 while 문의 조건식을 계속 만족하기 때문에 이 while 문은 무한 반복문이 됨

3. do - while 문

1) 사용 형식

```
do
{
    반복할 명령문
    조건식의 변수 변화
} while (조건식)
```

- 조건식을 만족하는 동안 do-while 문 안의 명령문들을 반복 실행
- 대부분 조건식에서 사용하는 변수에 대한 변화식을 반복하는 명령문에 포함
- 반복문을 한번 실행한 후에 조건식이 참(true)인지 판단

2) 사용 예

```
int j=1;
do {
    System.out.println(j);
    j++;
}while(j>10);
```

- 변수 j 의 값이 10 보다 큰 동안 j 를 출력하고 j 를 1 만큼 증가
- 조건식을 만족하지 않으므로 반복 종료
  ➔ 실행하면 10 보다 작은 짝수 출력
-

4. 반복문 제어 - break 문

1) 역할

- 실행 중인 반복 루프를 벗어남
- 반복문의 실행을 강제 종료

2) 사용 예

```
int i=0;
while(true) {
    System.out.println(i);
    i++;
    if(i>5)
        break;
}
```

- while 문의 조건식을 true 로 지정하여 while 문을 무한 반복문으로 생성
- 변수 i 를 1 씩 증가하며 출력하다가 i 값이 5 보다 커지는 순간에 break 문을 실행하여 while 문의 반복 종료

5. 반복문 제어 - continue 문

1) 역할

- 실행 중인 반복 루프는 유지
- continue 문 아래에 있는 반복문 안의 명령문은 실행하지 않고, 다음 번 반복을 실행

2) 사용 예

```
for(int j=0;j<10;j++) {
    if(j%2==0)
        continue;
    System.out.println(j);
}
```

- i 를 2 로 나눈 나머지가 0 과 같으면 continue 문을 실행하면서 그 아래에 있는 출력문을 실행하지 않고 for 문으로 돌아가 다음 번 반복을 진행
- 예시 코드를 실행하면 1 부터 10 까지의 수 중 홀수만 출력

3 API 클래스

1. String 클래스

1) 특징

- String 클래스는 문자열을 표현하기 위해 자바가 제공하는 클래스(java.lang 패키지에 포함됨)
- 문자열 변수의 선언 형식
  - ✓ String 변수명 = "문자열";

  예시)

```
String msg1 = "Hello_";
String msg2 = "1234";
```

2) + 연산

- "문자열 1" + "문자열 2" → "문자열 1 문자열 2"

```
String msg1 = "Hello_";
String msg2 = "1234";

System.out.println(msg1 + msg2);
```

〈 결과 〉
Hello_1234

- "문자열 1" + 기본 자료형(int, double, …) → "문자열 1 기본 자료형 데이터"

```
String msg1 = "Hello_";

System.out.println(msg1 + 1.0);
```

〈 결과 〉
Hello_1.0

3) String 클래스의 주요 메소드

(1) String 클래스(1) 예시

메소드 명	기능
char charAt(int index)	문자열에서 해당 인덱스 위치에 있는 문자 리턴
boolean isEmpty()	문자열이 비어 있으면 true
int length()	문자열의 길이 리턴
char[] toCharArray()	문자열을 char 배열로 반환

```
String str = "abcd";
System.out.println("str.charAt(2):" + str.charAt(2));
System.out.println("str.isEmpty():" + str.isEmpty());
System.out.println("str.length():" + str.length());

char[] charArr = str.toCharArray();
System.out.print("str.toCharArray(): ");
System.out.println(charArr);
System.out.println(charArr[0]);
//Arrays.toString()은 배열을 문자열로 반환
```

```
str.charAt(2):c
str.isEmpty():false
str.length():4
```

```
str.toCharArray(): abcd
a
```

(2) String 클래스(2) 예시

메소드 명	기능
String replace(char oldChar, char newChar)	oldChar 를 newChar 로 대체
boolean startsWith(String prefix)	시작 문자열이 prefix 와 동일하면 true
boolean endsWith(String suffix)	끝 문자열이 suffix 와 동일하면 true
boolean equals(Object anObject)	문자열과 인수로 받은 문자열의 동일 여부를 리턴

```
String str = "abcd";
```

```
System.out.println("str.replace():" + str.replace("bc","goo"));

System.out.println("str.startsWith():" + str.startsWith("a"));
System.out.println("str.endsWith():" + str.endsWith("a"));

System.out.println("str.equals():" + str.equals("good"));
```

```
str.replace():agood
str.startsWith():true
str.endsWith():false
str.equals():false
```

(3) String 클래스(3) 예시

메소드 명	기능
int indexOf(String str)	해당 문자열이 들어 있는 인덱스 번호 리턴
static String valueOf(int i)	값을 문자열로 변환하여 리턴

```
String str2 = "abcd";
int i = str2.indexOf("cd");
System.out.println("str2.indexOf(cd):"+ i);

i = str2.indexOf("c");
System.out.println("str2.indexOf(c):"+ i);

System.out.println("Strint.valueOf():"+ String.valueOf(3) );
```

```
str2.indexOf(cd):2
str2.indexOf(c):2
Strint.valueOf():3
```

(4) String 클래스(4) 예시

메소드 명	기능
String substring(int beginIndex, int endIndex)	시작 인덱스에서 끝 인덱스 전까지 부분 문자를 잘라 리턴. 끝 인덱스 생략 가능
String[] split(구분자)	구분자를 기준으로 문자를 잘라 String[]에 순서대로 리턴

```
String str3="123+12";
System.out.println("substring()-1:"+ str3.substring(2));
System.out.println("substring()-2:"+ str3.substring(1,4));

String str4="123,12,564";
String[] splitStr = str4.split(",");
System.out.println("split()-1:"+splitStr[0]);
System.out.println("split()-2:"+splitStr[1]);
```

```
substring()-1:3+12
substring()-2:23+
split()-1:123
split()-2:12
```

4) String 클래스 - equals() 더 알아보기

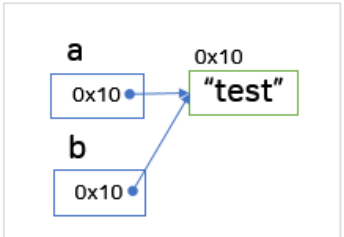
```
String a = "test";        String c = new String("test");
String b = "test";        String d = new String("test");
```

(1) == **연산자**

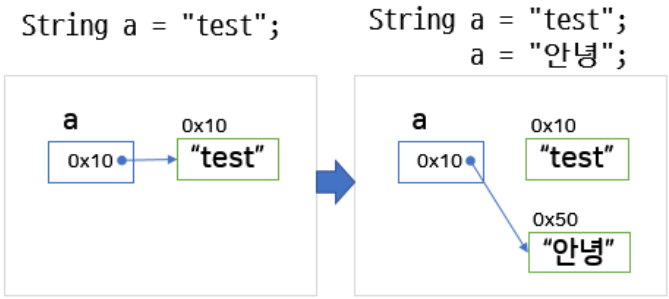
: 기본 자료형(primitive type)에서는 값을 비교하고, 참조 변수(reference type)에서는 주소 비교하여 같으면 true, 다르면 false 를 반환

변수 a 는 String 타입으로 참조 변수이므로 주소값을 가진다. 그러나 “test”라는 문자열 데이터의 값이 동일하므로 “test” 객체의 주소를 a 와 b 변수가 동시에 가리킴

```
System.out.println(a == b); //true
System.out.println(c == d); //false
```



만일 a 라는 변수에 “안녕”이라는 다른 데이터를 넣은 경우 a 안의 값을 바꾸는 것이 아님. 객체에서는 값을 바꾸는 것이 아니라 “안녕”이라는 객체를 새로 생성하여 “안녕”의 주소값을 새로 할당 받음



(2) **equals() 메소드**

: equals() 메소드는 객체 안의 데이터 값을 비교

equals() 메소드는 a.equals(b) 혹은 a.equals(“test”)처럼 두 인스턴스 변수 혹은 인스턴스 변수와 데이터 값이 같은지 비교하여 같으면 true, 다르면 false 를 반환한다.

```
System.out.println(a.equals(b)); //true
System.out.println(c.equals(d)); //true
```

※ String 인스턴스는 immutable 이다. 즉 한번 생성되면 값을 변경할 수 없으며, 값을 바꾸는 경우 새로 생성해서 다른 참조를 한다.

※ 문자열 덧셈의 경우 기존 문자열의 내용이 바뀌는 것이 아니라 내용이 합쳐진 새로운 String 인스턴스가 생성된다.

※ **문자열 토큰 구분하기**

StringTokenizer 클래스를 활용하여 특정 구분자를 기준으로 하여 문자열을 잘라낼 수 있다. String 클래스의 split() 메소드와 유사

```
import java.util.StringTokenizer;

public class ExamStringToken {

    public static void main(String[] args) {
        StringTokenizer st = new StringTokenizer("12:25:30",":");

        while (st.hasMoreTokens()) {
            System.out.println(st.nextToken());
        }
    }
}
```

2. Wrapper 클래스

자바의 기본 자료형과 매치되는 객체 타입
정수, 실수, 문자 등의 기본 자료형과 동일한 형태의 값을 가지나 객체 형태로 이루어진 데이터 타입

기본 타입	래퍼 클래스 (객체)
int	Integer
long	Long
float	Float
double	Double
char	Character
boolean	Boolean

박싱(Boxing)은 기본 타입을 래퍼 클래스로 전환하는 것을 말하고, 언박싱(Unboxing)은 래퍼 클래스를 기본 타입으로 변경하는 것을 의미한다.

박싱(Boxing)	언박싱(Unboxing)
기본 타입 ➔ 래퍼 클래스로 전환	래퍼 클래스 ➔ 기본 타입으로 전환
`Integer i = new Integer(100);`	`int ui = i.intvalue();`

그러나, JDK 1.5 이후 버전부터는 **Auto Boxing, Unboxing** 되어 특별히 변환에 신경 쓰지 않아도 자동으로 변환된다.

박싱(Boxing)	언박싱(Unboxing)
`Integer i = 100;`	`int ui = i;`

단, 데이터가 문자열인 경우는 아래와 같이 변환해야 한다.

박싱(Boxing) Wrapper 클래스로 변환	언박싱(Unboxing) 기본형으로 변환하는 경우
`Integer i = Integer.valueOf("100");` `Float f = Float.valueOf("10.1");`	`int i = Integer.parseInt("100");` `float f = Float.parseFloat("10.1");`