Professional Coding Specialist

COS Pro JAVA 1 급

4 강. 기초 문법 정리 4

1. API 클래스(2)
2. 컬렉션 프레임웍
3. 제네릭

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 네 번째 시간으로, 자주 사용하는 API 클래스에 더 알아보고, 컬렉션 프레임웍, 제네릭에 대한 개념을 정리한다.

학습 목차

1. API 클래스(2)
2. 컬렉션 프레임웍
3. 제네릭

학습 목표

1. 자바에서 자주 사용하는 API 클래스를 정리한다.
2. 자바의 컬렉션 프레임웍의 개념과 사용법에 대해 정리한다.
3. 자바의 제네릭의 개념과 사용법에 대해 정리한다.

1 API 클래스(2)

1. Math 클래스

1) 특징

- 수학적으로 자주 사용하는 함수를 구현해 놓은 클래스
- java.lang 패키지에 포함되어 있으므로 별도의 import 를 할 필요가 없음

2) Math 클래스의 주요 메소드

메소드 명	기능
Math.random( )	0.0 이상 1.0 미만의 임의의 실수값을 리턴
Math.abs(값)	절대값을 리턴
Math.max(값 1, 값 2)	값 1 과 값 2 중 더 큰 값을 리턴
Math.min(값 1, 값 2)	값 1 과 값 2 중 더 작은 값을 리턴
Math.ceil(값)	소수 부분을 올린 값을 리턴
Math.floor(값)	소수 부분을 버린 값을 리턴
Math.round(값)	소수 첫째 자리에서 반올림한 값을 리턴
Math.sqrt(값)	제곱근 값을 리턴

```
for (int i=0 ; i<10; i++)
    System.out.println( Math.random() );
```

```
0.8310775389161198
0.04264265609984608
0.036525086767813075
```

```
for (int i=0 ; i<10; i++)
    System.out.println((int)( Math.random()*100 ) );
```

```
28
22
6
```

```
int a = 10;
int b = -6;
System.out.println(Math.abs(b));
System.out.println(Math.max(a, b));
System.out.println(Math.round(1.6));
```

```
8
6
10
2
```

2. Arrays 클래스

1) 특징

- 배열을 다루기 위해 자바가 제공하는 클래스
- java.util 패키지에 포함되어 있기 때문에 Arrays 클래스의 메소드를 사용하려면 java.util 패키지를 임포트 해야 함

2) Arrays 클래스의 주요 메소드

- Array 클래스의 메소드들은 모두 static 메소드이므로 Arrays 클래스에 대한 객체를 생성하지 않고 사용하기 때문에 사용할 때 'Arrays.메소드이름( )' 으로 실행

(1) Arrays.fill( ) & Arrays.toString( ) 메소드

구분	메소드 명	기능
값 초기화	Arrays.fill(배열명, 값)	배열의 값 초기화
배열 출력	Arrays.toString(배열명)	배열의 항목 전체를 한 줄의 문자열로 변환하여 리턴

- **Arrays.fill( ) 메소드**
  - 배열 항목의 값을 초기화 하는 메소드
  - Arrays.fill(배열명, 값)의 형태로 사용

```
int[] arrB = new int[8];

Arrays.fill(arrB, 3);
System.out.println("Arrays.fill(): " + Arrays.toString(arrB));

Arrays.fill(arrB, 0);
```

```
Arrays.fill(): [3, 3, 3, 3, 3, 3, 3, 3]
```

- **Arrays.toString( ) 메소드**
  - 배열 항목을 한 번에 한 줄의 문자열로 출력하는 메소드

```
int a[] = new int[] {3,5,7,9};
System.out.println(Arrays.toString(a));
```

```
[3, 5, 7, 9]
```

(2) 배열 복사 메소드

패키지 명	메소드 명	기능
Java.util	int[] copyOf(int[] original, int newLength)	배열에서 해당 길이를 갖는 새 배열로 복사해서 리턴
	int[] copyOfRange(int[] original, int from, int to)	배열의 from 에서 to 이전 인덱스까지 복사해서 리턴

Java.lang	void arraycopy(Object src, int srcPos, Object dest, int destPos, int length) - System 클래스	src 배열의 srcPos 인덱스부터 시작해서 총 length 까지 복사 (dest 배열의 destPos 인덱스부터) arraycopy 는 dest 배열이 미리 존재해야 함

- **Arrays.copyOf( ) 메소드**
  - 배열에서 해당 길이를 갖는 새 배열로 복사해서 리턴
  - Arrays.copyOf(배열명, 길이)의 형태로 사용
- **Arrays.copyOfRange( ) 메소드**
  - 배열의 from 에서 to 이전 인덱스까지 복사해서 리턴
  - Arrays.copyOfRange(배열명, 시작위치, 끝위치-1)의 형태로 사용
- **System.arraycopy( ) 메소드**
  - src 배열의 srcPos 인덱스부터 시작해서 총 length 까지 복사 (dest 배열의 destPos 인덱스부터)
  - java.lang 패키지의 System 클래스에서 제공
  - System.arraycopy(src 배열, dest 배열, dest 시작위치, 길이)의 형태로 사용

```
int[] arrA = {5,9,2,1,3,6,7,3};
int[] arrB = new int[8];
```

```
//Arrays.copyOf()
int[] arcopyOf = Arrays.copyOf(arrA, 2);
System.out.println("Arrays.copyof(): " + Arrays.toString(arcopyOf));

//Arrays.copyofRange()
int[] arcopyofRange = Arrays.copyOfRange(arrA, 1, 5);
System.out.println("Arrays.copyofRange(): " +Arrays.toString(arcopyofRange));

//System.arraycopy()
System.arraycopy(arrA, 1, arrB, 0, 3);
System.out.println("System.arraycopy()-일부: " + Arrays.toString(arrB));
System.arraycopy(arrA, 0, arrB, 0, arrA.length);
System.out.println("System.arraycopy()-전부: " + Arrays.toString(arrB));
```

```
Arrays.copyof(): [5, 9]
Arrays.copyofRange(): [9, 2, 1, 3]
System.arraycopy()-일부: [9, 2, 1, 0, 0, 0, 0, 0]
System.arraycopy()-전부: [5, 9, 2, 1, 3, 6, 7, 3]
```

(3) Arrays.sort() & Arrays.equals() & Arrays.binarySearch() 메소드

패키지 명	메소드 명	기능
정렬	Arrays.sort(배열명)	배열을 오름차순으로 정렬
배열 비교	boolean equals(int[] a, int[] b)	두 배열의 내용을 비교해서 true / false 리턴

탐색	Arrays.binarySearch(int[] a, int key)	배열에서 key 를 찾아서 있으면 해당 인덱스 값 리턴 (없으면 0 보다 작은 수) 주의 : 이진 탐색 알고리즘을 사용하므로 반드시 배열이 정렬되어 있어야 함

- **Arrays.sort( ) 메소드**
  - 배열을 오름차순으로 정렬
  - Arrays.sort(배열명)의 형태로 사용
- **Arrays.equals( ) 메소드**
  - 두 배열의 내용을 비교해서 true / false 리턴
  - Arrays.equals(배열명, 배열명)의 형태로 사용
- **System.binarySearch( ) 메소드**
  - 배열에서 key 를 찾아서 있으면 해당 인덱스 값 리턴 (없으면 0 보다 작은 수)
  - 이진 탐색 알고리즘을 사용하므로 반드시 배열이 정렬되어 있어야 함
  - Arrays.binarySearch(배열명, key 값)의 형태로 사용

```
int[] arrA = {5,9,2,1,3,6,7,3};
int[] arrB = new int[8];
```

```
//Arrays.sort()
Arrays.sort(arrA);
System.out.println("Arrays.sort()-정렬: " + Arrays.toString(arrA));

//Arrays.equals()
System.out.println(Arrays.equals(arrA,arrB));
arrB = Arrays.copyOf(arrA,arrA.length);
System.out.println("Arrays.equals(): " + Arrays.equals(arrA,arrB));

//Arrays.binarySearch() 배열이 정렬되어 있어야 함
System.out.println("Arrays.binarySearch(): " +Arrays.binarySearch(arrA, 9));
```

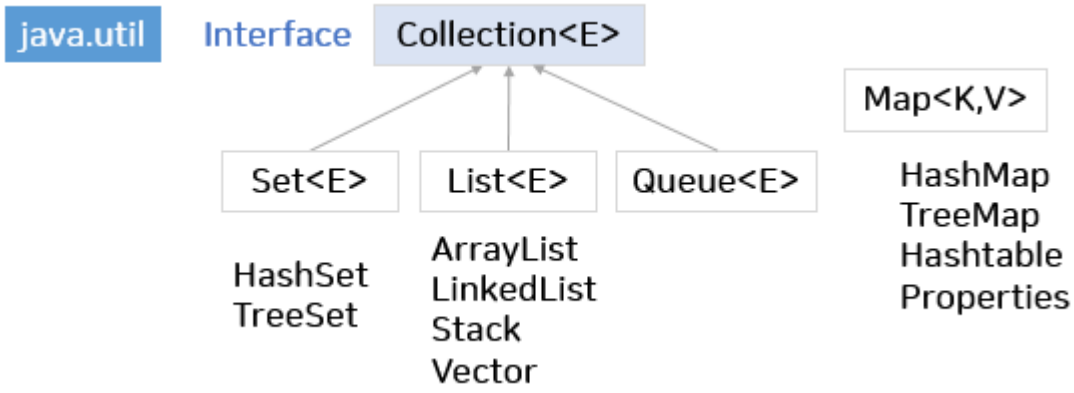
```
Arrays.sort()-정렬: [1, 2, 3, 3, 5, 6, 7, 9]
```

```
false
Arrays.equals(): true
Arrays.binarySearch(): 7
```

2 컬렉션 프레임웍(Collection Framework)

1. 컬렉션 프레임웍 정의

- 잘 정의된 구조의 클래스 모음
- 제네릭 기반으로 구현되어 있음
- 개발 생산성을 높이고 유지보수를 용이하게 함
- 데이터 저장 방식 자료 구조 : 리스트, 스택, 큐, 트리, 해쉬
- 활용 알고리즘 : 버블 정렬, 퀵 정렬, 이진 탐색...



2. List<E>

- 컬렉션 프레임웍 중 하나
- 인터페이스로 리스트 자료구조의 특징인 중복 허용, 저장 순서 유지되는 컬렉션을 구현하기 위해 사용
- ArrayList, LinkedList, Stack, Vector 등의 자료 구조

- **리스트 자료구조에서 사용할 수 있는 메소드**

메소드 명	기능
void add(int index, Object element) boolean addAll(int index, Collection c)	지정된 index 에 해당 element 추가
Object get(int index)	지정된 index 의 객체 얻기
Object set(int index)	지정된 index 의 객체 저장하기
ListIterator listiterator()	listiterator 를 얻음
boolean remove(int index)	객체 삭제

1) ArrayList

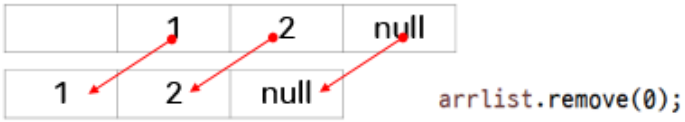
ArrayList 는 배열 구조와 동일. 다만 여러 가지 메소드를 제공하기 때문에 다양한 기능들을 쉽게 이용할 수 있다는 장점이 있음

- 데이터 읽기
  - 인덱스 번호로 데이터를 읽어 들이기 때문에 쉽고 빠르게 데이터를 찾아와서 읽어 들일 수 있음
  - 속도가 빠르고 효율적

인덱스번호	0	1
데이터	1	2

- 삽입
  - 데이터 삽입 시 삽입하고자 하는 인덱스 위치의 뒤에 있는 모든 데이터를 하나씩 뒤로 변경
  - 속도가 저하되고 비효율적

- 삭제
  - 삭제하고자 하는 데이터를 삭제 후 삭제된 인덱스 위치의 뒤에 있는 모든 데이터를 하나씩 앞으로 변경
  - 속도가 저하되고 비효율적



- **ArrayList 사용 예**

  ArrayList 를 Integer 타입으로 생성해서 값을 하나씩 추가하고, 그 사이즈만큼 돌면서 i 번째 있는 값을 get 얻어와서 출력하고 삭제하는 예제

```java
ArrayList<Integer> arrlist = new ArrayList<>();

arrlist.add(1);
arrlist.add(2);
arrlist.add(0,10);

System.out.println(arrlist);                    // 10 1 2

for(int i=0; i<arrlist.size();i++) {
    System.out.print(arrlist.get(i)+"  ");      // 10 1 2
}
System.out.println();

arrlist.remove(0);

for(int i=0; i<arrlist.size();i++) {
    System.out.print(arrlist.get(i)+"  ");      // 1 2
}
```

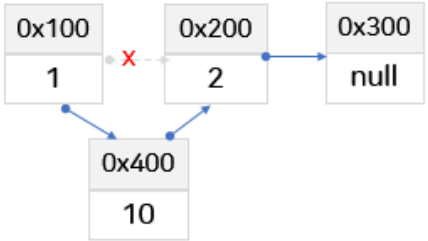
2) LinkedList

LinkedLis 는 데이터들이 사슬처럼 체인 구조로 연결되어 있음. 데이터 값과 함께 앞뒤 데이터의 주소값을 같이 가지고 있는 구조로 새로운 데이터 삽입과 삭제 시 링크만 변경하면 되는 장점이 있음

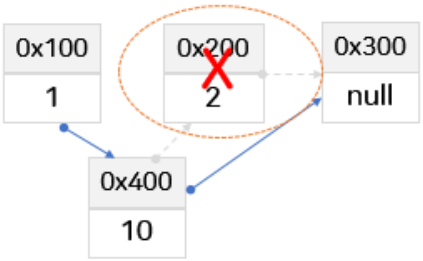
이전	데이터	다음
0x100	2	0x300

- 데이터 읽기
  - 데이터와 데이터의 연결을 계속 따라가야 하는 형태
  - 속도 느리고 비효율적

- 삽입
  - 데이터 삽입 시 삽입하고자 하는 곳의 위치 연결을 끊고 새로운 데이터의 주소만 연결해 주는 형태
  - 속도가 빠르고 효율적



- 삭제
  - 데이터 삭제 시 삭제하고자 하는 곳의 위치 연결을 끊고 다음 데이터 주소만 연결해 주는 형태
  - 속도가 빠르고 효율적



- **LinkedList 사용 예**
  LinkedList 를 Integer 타입으로 생성해서 값을 하나씩 추가하고, 그 사이즈만큼 돌면서 i 번째 있는 값을 get 얻어와서 출력하고 삭제하는 예제

```
LinkedList<Integer> lnklist = new LinkedList<>();

lnklist.add(1);
lnklist.add(2);
lnklist.add(0,10);

System.out.println(lnklist);

for(int i=0; i<lnklist.size();i++) {
    System.out.print(lnklist.get(i)+"  ");
}
System.out.println();

lnklist.remove(0);

for(int i=0; i<lnklist.size();i++) {
    System.out.print(lnklist.get(i)+"  ");
}
```

```
[10, 1, 2]
```

```
10  1  2
```

```
1  2
```

※ **ArrayList 와 LinkedList 는 결과값은 동일하나 안에서 동작하는 형태가 다름**

구분	데이터 읽기	추가, 삭제
ArrayList	빠름	느림 (데이터 개수가 변하지 않는 경우 효율적)
LinkedList	느림 (데이터가 많으면 특정 데이터를 찾는데 오래 걸림)	빠름 (데이터 개수가 많이 변하면 효율적)

3 제네릭(Generic)

1. 제네릭 정의

- 클래스 작성 시 자료형에 의존적이지 않은 클래스를 정의
- 인스턴스 생성 시 T 의 자료형을 결정하는 것이 제네릭
- 특정한 타입 대신 타입 변수(E or T ...)를 선언

```
class GenClass<T> { ... }


GenClass<String> var1 = new GenClass<String>();
GenClass<Integer> var2 = new GenClass<Integer>();
```

```
ArrayList<Integer> var3 = new ArrayList<Integer>();
```

1) 제네릭 사용 예

- **JDK 1.5 버전 이전의 제네릭 사용 예**
  - <>안에 데이터 타입이 없음

```
ArrayList arrlist= new ArrayList();

arrlist.add(1);
arrlist.add(2);
arrlist.add("3");

System.out.println(arrlist);
```

```
ArrayList arrlist= new ArrayList();

arrlist.add(1);
arrlist.add(2);
arrlist.add(3);
Integer var1 = (Integer)arrlist.get(2);
```

- **JDK 1.5 버전 이후의 제네릭 사용 예**
  - jdk1.5 이후부터는 ArrayList 가 제네릭으로 변경
  - 반드시 <> 타입을 표기해야 함
  - 컴파일러에게 타입을 미리 알려주어 타입 체크 강화(형변환 에러 줄임)

```
ArrayList<Integer> var3 = new ArrayList<Integer>();
```

```
ArrayList<Integer> arrlist= new ArrayList<Integer>();

arrlist.add(1);
arrlist.add(2);
arrlist.add("3");  //컴파일 에러

System.out.println(arrlist);
```

```
ArrayList<Integer> arrlist= new ArrayList<Integer>();

arrlist.add(1);
arrlist.add(2);
arrlist.add(3);
Integer var2 = arrlist.get(2);
```

2. Iterator

컬렉션에 저장된 요소를 접근하는데 사용되는 인터페이스

1) 사용 형식

```
LinkedList<Integer> lnklist = new LinkedList<>();

lnklist.add(1);
lnklist.add(2);
lnklist.add(0,10);

Iterator<Integer> iter = lnklist.iterator();
Integer v1;
while(iter.hasNext()) {
    v1=iter.next();
    if (v1==1)
        iter.remove();
}

System.out.println(lnklist);                [10, 2]
```

메소드	설명
E next()	다음 요소 반환
boolean hasNext()	다음 요소가 있는지 여부
E previous()	이전 요소 반환
boolean hasPrevious()	이전 요소가 있는지 여부