7.기본적인 클래스 구성방법

1) 캡슐화(Encapsulation) 및 클래스 형식

캡슐화(Encapsulation)란 관계된 목적을 가지는 변수와 함수들을 하나의 클래스로 만들어 외부에서 쉽게 데이터에 접근하지 못하도록 제한하고 은닉하는것이다. 캡슐화의 가장 큰 이유는 외부에서의 정보접근이나 제한을 두기 위함이며 변경할수 없게 직접적인 접근을 막고 해당 클래스가 정의한 데이터와 메소드를 통해서만 접근이 가능하다.

다음은 클래스 파일의 보편적인 구성(캡슐화) 형식이다.

1. 패키지명
2. 클래스명 //30자 이내로.
3. 데이터(인스턴스 변수 == 멤버 ==필드):
4. 생성자 함수
5. 메소드(함수)
6. getter & setter

Ex) 클래스 구성 형식으로 만든 예제

```
 1 package com.lec.ex1_square; //패키지 명
 2
 3 public class Square {       //클래스 선언
 4     private int side;       //데이터 선언
 5
 6     public Square() {       //파라미터 없는 생성자 함수 선언
 7         System.out.println("매개변수 없는 생성자 함수 호출됨");
 8     }
 9     public Square(int side) {   //1개의 파라미터를 가지는 생성자 함수 선언
10         this.side = side;
11         System.out.println("매개변수 있는 생성자 함수 호출. side 초기화");
12     }
13     public int area() {         //넓이를 구하고 넓이값을 리턴하는 매서드 선언
14         return side * side;
15     }
16
17     public void setSide(int side) { //데이터 setter 선언
18         this.side = side;
19     }
20     public int getSide() {          //데이터 getter 선언
21         return side;
22     }
23
```

2)생성자 함수

생성자의 기본 목적은 객체를 생성함과 동시에 초기화 하려는 데있다.

생성자 함수의 특징으로는

- 리턴 타입이 없으며 해당 클래스와 같은 이름으로 생성되야하며
- 모든 클래스는 반드시 하나 이상의 생성자가 있어야 한다 ,
- 만약 하나도 없으면 JVM(Java Virtual Machine)이 매개변수가 없는 디폴트(default) 생성자를 만들어 준다.
- 접근제어자는 public이어야한다.
- 자바에서 매개변수가 없는 디폴트 생성자를 생성해버린다.

3)this

자신이 상속받는 클래스에서의 변수나 지역변수가 아닌 상위변수를 명시할 때 ,

광역변수와 지역변수의 이름이 같을 때 구분, 명시 해준다.

Ex)

```
1 public class Member {//클래스 아래 선언된 광역변수
2     private String id;
3     private String pw;
4 
5     public Member(String id, String pw){//매개변수로 들어오는 이름이 같은 지역변수
6         this.id = id;   //this.변수명 은 클래스 아래에 선언된 광역변수를 가르킨다.
7         this.pw = pw;
8 
9   }
```

3)Import

Import란 다른 패키지의 클래스를 가져와 사용할 때 패키지명을 명시하여 사용하여하 하지만,

import를 통해 클래스의 패키지에 대한 정보를 가져와 패키지명을 생략하고 사용할수있다..

모든 소스파일에서 패키지문 다음, 클라스문 이전으로 선언한다.

Ex)

```
1  //import.패키지파일명.*는
2  //com.lec.ex2_human.해당 패키지의 하위 모든 class를 import한다.
3 
4 import com.lec.ex2_human.*;
5 
6  //단축키 ctrl shift O 를 통해 해당 클래스에서 사용하지않는 클래스들은 import 삭제
7  //수많은 클래스 사이에서 여러개를 써야한다면 모든 클래스를 동기화 해놓고
8  //사용 후 안쓰는 클래스를 동기화 삭제하는것이 효율적이다.
```

4).format

```
1 int i = 23;
2 
3 System.out.println(String.format("%d_", i));
4 System.out.println(String.format("%5d_", i));
5 System.out.println(String.format("%-5d_", i));
6 System.out.println(String.format("%05d_", i));
7 
8 23_
9     23_
10 23    _
11 00023_
```

.format문을 통해 문자열을 사용한 형식 문자열을 만들 수 있다

- %5d 와 같이 %와 d 사이에 정수를 설정하면, 글자 길이를 설정할 수 있다.
- 기본적으로 오른쪽 정렬이고, -를 붙일 경우 왼쪽정렬된다.(ln 4~5)
- % 바로 뒤에 , 를 붙이면 3자리 단위로 쉼표를 표시해준다.

5) .equal 문

.equal문은 문자열을 비교하는 매소드이다.

- ==연산자와 equals()메소드의 가장 큰 차이점은 == 연산자는 비교하고자 하는 대상의 주소값을 비교하는데 반해 String클래스의 equals 메소드는 비교하고자 하는 대상의 값 자체를 비교한다는 것이다.

Ex)

```
1  String s1 = "abcd";
2  String s2 = new String("abcd");
3
4   if(s1 == s2) { // == 연산자로 비교 //가르키는 주소를 비교함
5        System.out.println("두개의 값이 같습니다.");
6    }else {
7        System.out.println("두개의 값이 같지 않습니다.");
8    }
9 //두개의 값이 같지 않습니다.
```

== 연산자의 경우 참조 타입 변수들 간의 연산은 동일한 객체를 참조하는지, 다른 객체를 참조하는지 알아볼 때 사용되며, 참조 타입의 변수의 값은 힙 영역의 객체 주소이므로 결국 주소 값을 비교하는 것이 되어 다르다는 결론이 나온다. 그래서 자바에서 문자열을 비교하려면 equals라는 메서드를 활용하여 비교해야한다.

```
1    String s1 = "abcd";
2    String s2 = new String("abcd");
3
4     if(s1.equals(s2)) { //.equals연산자를 이용한 문자열 비교
5        System.out.println("두개의 값이 같습니다.");
6     }else {
7        System.out.println("두개의 값이 같지 않습니다.");
8     }
9 //두개의 값이 같습니다.
```

반면 String 클래스안에 있는 equals라는 메서드를 사용하면 두 비교대상의 주소 값이 아닌 데이터값을 비교하기 때문에 어떻게 String을 생성하느냐에 따라 결과가 달라지지 않고 정확한 비교를 할 수 있다.