6. 객체지향 프로그래밍이란

먼저 기존의 절차지향 프로그래밍의 경우 동일한 로직을 반복하려면 다시 코딩하거나 복사해야하는데 이 경우 코드의 길이가 길어져 비효율적이며 알아보기도 힘들다. 그러므로 함수(메소드)를 만들어 해당 로직을 메소드로 활용하는 것이 효율적으로 생각하였다. 하지만 메소드를 이용한 방식에도 한계가 있었으며 데이터와 메소드가 많아짐에 따라 같은 부류의 데이터와 메소드를 분류하고 모아놓고 필요에 따라 사용하게 되었는데 이렇게 모아놓은 덩어리를 객체라고한다.

1) 메소드 (Method) == 함수

작업을 수행하기 위한 명령문의 집합. 어떤 값을 입력 받아서 처리하고 그 결과를 돌려준다 (입력 받는 값이 없을 수도 있고ex)매소드명(); 결과를 돌려주지 않을 수도 있다. (void형태)) 메소드 클래스(Class)에 있는 함수이며 자바에서 모든 함수는 클래스에 있기 때문에 사실 자바의 모든 함수는 메소드라고 할수있다.

2) 매소드 형태

```
Ex) public String evenOdd(int value) { // 접근제한자 [static] 메소드명[매개변수]
        String result = null;
        if(value%2==0)
                result = "짝수입니다";
        else
                result = "홀수입니다";
        return result;  // 결과값 리턴
    }
Ex2) int sum(int x, int y) {   // int 라는 자료형을 갖는 sum이라는 이름의 메소드
    return x + y; }        //입력값(매개변수 == 파라미터)을 받아(int x, int y 결과를 리턴하는 함수
```

- 메소드의 접근 제한자(ex. public) 에 따라 메소드의 활용범위가 제한된다

제어자	같은 클래스	같은 패키지	자손클래스	전 체
public	○	○	○	○
protected	○	○	○	
(default)	○	○		
private	○			

● 또한 메소드의 리턴타입 앞에 static이라는 형식의 유무에 따라 활용방법이 달라진다. Static이 붙은 함수는 이클립스상에 미세하게 기울여져 표시되며 , static키워드를 사용한 변수는 클래스가 메모리에 자동으로 올라서 인스턴트(객체) 없이 바로 사용 가능하다.

Ex) public class Car //Car라는 클라스에 선언된

public void drive() //static 이 없는 drive()매서드는 인스턴트를 만들어 사용해야한다.

System.*out*.println("운전한다. 지금 속도 : "+speed);

}

Ex) Car c1 = new Car(); //Car클래스의 c1이라는 인스턴트를 생성

c1.drive(); //c1이라는 인스턴트 객체에 복사된 Car의 drive()를 호출하여 사용

Ex) public class Car //Car라는 클라스에 선언된

public static void drive() //static 이 있는 drive()매서드는.

System.out.println("운전한다. 지금 속도 : "+speed);

Ex) Car.drive() //이 같은 형태로 클라스.매서드명 으로 바로 호출 가능하다.

3) 메소드의 구분

메소드 == 함수는 매개변수를 받는데 이때 받는 매개변수의 개수나 데이터 타입으로 같은 이름의 함수를 재정의하여 구분해서 사용할수있다. 이를 오버로딩(overlording)이라한다.

```
12
13      private static void printLine(char c, int cnt) {
14          for (int i = 0; i < cnt; i++) {
15              System.out.print(c);
16          }
17          System.out.println();
18      }
19
20      private static void printLine(int cnt) {
21          for (int i = 0; i < cnt; i++) {
22              System.out.print('-');
23          }
24          System.out.println();
25      }
26
27      private static void printLine(char c) {
28          for (int i = 0; i < 30; i++) {
29              System.out.print(c);
30          }
31          System.out.println();
32      }
33
34      private static void printLine() {
35          for (int i = 0; i < 30; i++) {
36              System.out.print('-');
37          }
38          System.out.println();
```

- 매개변수와 데이터타입을 다르게 해서 정의(오버로딩)한 매소드 -

4) Setter & getter

자바(객체지향 언어)로 프로젝트를 진행하는 프로그램을 보면 변수의 접근제어자가 private로 설정되있는 것을 자주 볼 수 있다. 이는 외부에서의 무분별한 데이터 설정을 막고 오류를 검증하기 위함으로 만약 외부에서 수를 입력할 때 음수나 기대한 수의 범위를 벗어나 입력할 때를 검증하고 오류를 없애기 위함이다.

이때 setter는 set"대문자로 시작하는 변수명" 으로 짓는 것이 보편적이다.

- 재귀적 함수호출이란

해당 매소드 내부에서 또다시 매소드를 불러 반복적으로 작업하는 것을 말한다. 이러한 재귀 호출은 자기가 자신을 계속해서 호출하므로, 끝없이 반복되므로 메소드 내에 재귀 호출을 중단하도록 조건이 변경될 명령문을 반드시 포함해야 한다

Ex) 팩토리얼(!) 알고리즘

```
long factorial(int su1) {     // su가 1초과일경우: su * factorial(su-1)
    if (su1 == 1) {
      return 1;
            }else {
              return su1 * factorial(su1-1);     //함수 안에 함수 호출
```