Day 11

태그	
날짜	@2022년 10월 14일

객체지향 Object Oriented Programming

클래스, 객체, 인스턴스, 상속, 인터페이스, 다형성, 추상화 등의 개념을 포함한다.

- public 두개 사용 불가능
- 파일 하나당 클래스 하나가 좋다.

- exam - 계산기

```
package day11_1014;

class c1_Calculator1 {

  static int result = 0;
  static int add(int num) {

    result += num;
    return result;
  }

}

class c1_Calculator2 {

  static int result = 0;
  static int add(int num) {

    result += num;
    return result;
  }

}
public class c1_Calculator_Sample {

  public static void main(String[] args) {
    System.out.println(c1_Calculator1.add(3));
    System.out.println(c1_Calculator1.add(4));
    System.out.println();
    System.out.println(c1_Calculator2.add(1));
```

```
    System.out.println(c1_Calculator2.add(2));
  }

}
```

출력값 :

3
7

1
3

만약 계산기를 수십개로 늘려야 한다면, 또 그 수십개의 계산기의 기능을 수정해야 한다면 많은 시간이 든다.

- 위 코드 와 달리 하나의 클래스로 만든 객체 cal1, cal2… 들은 각기 다른 개성을 갖게되어 각각의 값을 유지하게 된다. 또한 추가 기능이 필요할 경우에도 하나의 클래스에만 적용하면 모든 객체에서 동일하게 그 기능을 사용할 수 있게 된다.

하나의 클래스로 만든 객체들은 각기 다른 값을 유지할 수 있다.

```
package day11_1014;

class c1_Calculator1 {

   int result = 0;
   int add(int num) {

    result += num;
    return result;
  }

}

public class c1_Calculator_Sample {

  public static void main(String[] args) {

    c1_Calculator1 cal1 = new c1_Calculator1(); // 다른 개성 의 cal1
    c1_Calculator1 cal2 = new c1_Calculator1(); // 또 다른 개성의 cal2

    System.out.println(cal1.add(3));
    System.out.println(cal1.add(4));
    System.out.println();
    System.out.println(cal2.add(1));
    System.out.println(cal2.add(2));
  }
```

```
}
```

객체와 인스턴스

- 객체와 인스턴스는 동일하다.
- 인스턴스는 클래스와의 관계를 설명할때 사용된다.

  예를 들면 위에서 cal1객체와 cal2객체는 Calculator 클래스로 찍어낸 객체이다. 라는 표현보다는 cal1객체와 cal2객체는 Calculator 클래스의 인스턴스이다. 라는 표현이 더욱 어울린다.

클래스와 객체

- 붕어빵 틀 = 클래스(class) , 붕어빵 = 객체,인스턴스

```
package day11_1014;

class Animal {

  String name; // 객체 변수, 인스턴스 변수, 멤버 변수

  public void setName(String name) {
    this.name = name;

    // this 이 매서드를 호출한 그 이름이 this에 오게됨
    // 예로 this 자리에 cat이 오게됨
  }
}









public class c2_Sample1 {

  public static void main(String[] args) {
    Animal cat = new Animal();
    // Animal은 자료형,Animal()은 생성자
    // new는 새로운 객체를 생성 할때 사용하는 키워드
    // cat 인스턴스는 Animal 클래스로 만들었다.
    // 붕어빵 틀 = 클래스 , 붕어빵 = 객체
```

```
    System.out.println(cat.name);
    cat.setName("bob");
    System.out.println(cat.name);

    System.out.println();

    Animal dog = new Animal();
    System.out.println(dog.name);
    dog.setName("hong");
    System.out.println(dog);
    System.out.println(dog.name);
    Animal horse = new Animal();

  }

}
```

결과를 확인 해보면 name 객체 변수 값은 공유되지 않고 각기 따로 유지되는 것을 확인할 수 있다.

이부분이 가장 중요한 객체지향의 개념이다.

객체들의 변수 값들이 독립적으로 유지되는 것이 바로 클래스의 존재의 이유이기도 하다.

(static을 사용하여 객체들간의 변수값을 공유하는 방법도 존재하기는 하지만 중요하지 않다.)

메서드 Method

다른언어에서는 함수라고 부르기도 하지만 자바에서는 모든 것이 클래스 안에 존재하기 때문에 함수라는 표현은 쓰지 않고 메서드라는 표현만 사용한다.

파이썬에선 클래스 밖에 메서드가 존재 → 함수라고 한다.

- 메서드를 사용하는 이유

  프로그래밍을 하다보면 똑같은 내용을 반복해서 처리해야 하는 일이 대부분이다.

  여러 번 반복해서 사용하는 기능을 체계적으로 한번만 구성 해놓으면 필요할 때마다 호출하여 편리하게 사용할 수 있다.

```
int sum(int a, int b) {

return a+b;
```

```
}

객체명.sum(2,3)
```

sum 메서드는 입력값으로 두개 a,b를 받아 return값으로 두개의 값을 더한 결과를 돌려준다.

```
package day11_1014;

public class c3_Method_Exam {


    int sum(int a, int b) {
      return a+b;
    }

    public static void main(String[] args) {
      c3_Method_Exam Sample = new c3_Method_Exam();

      System.out.println(Sample.sum(3,4));
    }
  }
```

- 위 코드는 sum 메서드에 3, 4를 전달하여 7 이라는 값을 돌려 받게 된다.

  7이 출력되는 것을 확인 할 수 있다.

매개변수와 인수

매개변수는 parameter 인수는 arguments 역시 혼용해서 사용되는 헷갈리는 용어

매개변수는 메서드에 입력되는 전달 값을 받는 변수를 의미하고 인수는 메서드를 호출할 때 전달 되는 값을 의미

```
    int sum(int a, int b) {
      return a+b;
    }
```

위 코드에서 sum 메서드의 int a, int b를 매개변수라고 부르고

```
System.out.println(Sample.sum(3,4));
```

매서드 호출 구문에서의 3, 4 를 인수라고 부른다.

(진짜 들어가는 실제 값)

- 기본형태

```
int(리턴 자료형) sum(메서드 이름) (int a, int b ....)(매개변수)

        return  리턴값;

int sum(int a, int b) {
      return a+b;
}
```

- **메서드는 크게 4가지가 존재한다.**
  - 입력과 출력이 모두 있는 메서드 (가장 많이 쓰는 형태의 메서드)
  - 입력과 출력이 모두 없는 메서드
  - 입력은 없는데 출력은 있는 메서드
  - 입력은 있는데 출력은 없는 메서드

```
입출력이 모드 있는 메서드
- 기본형태
int sum(int a, int b){
  return a+b;
}

----------------------------------------------------------------------

package day11_1014;
```

```
public class c3_method_type_exam {


  // 입출력이 모두 있는 메서드
  int inputout(int a, int b) {
    return a*b;
  }
  public static void main(String[] args) {

    c3_method_type_exam test = new c3_method_type_exam();
    System.out.println(test.inputout(2, 5));

  }

}

출력값 : 10
```

```
입출력이 모두 없는 메서드
- 기본형태
void nomethod(){
 System.out.println("인쇄");
}

// 메서드가 없을 뿐이지만 호출만 하면 무조건 찍어냄.
// return이 없으면 돌가서 출력하는것이 없는 것이다.

------------------------------------------------------------
package day11_1014;

public class c3_method_type_exam {

void noinout() {
    System.out.println("환영합니다.");
    }

public static void main(String[] args) {

test.noinout();

출력값 : 환영합니다.
```

```
입력은 없고 출력은 있는 메서드
- 기본 형태
int noinput(){
    return 7;
}
```

```
// 그냥 무조건 7이 가버리는

------------------------------------------------------------
package day11_1014;

public class c3_method_type_exam {

int noinput() {
    return 15;
  }

public static void main(String[] args) {

System.out.println(test.noinput());
  }

}

출력값 : 15
```

```
입력은 있는데 출력은 없는 메서드
- 기본형태
void yesinput(String a);
    System.out.println(a + "님 환영합니다.");
}

------------------------------------------------------------------------------
package day11_1014;

public class c3_method_type_exam {

void yesinput(String name) {
    System.out.println(name + "님 환영합니다.");
  }

public static void main(String[] args) {

  test.yesinput("이민호");

}

}

출력값 : 이민호님 환영합니다.
```

return의 또 다른 용도

return 메서드를 강제로 빠져나가는 용도로도 사용된다.

```
package day11_1014;

public class c3_other_ruturn {

  void sayNick(String nick) {
    if("fool".equals(nick)) { //nick 이랑 fool이랑 같니
      return; // nick과 "fool" 이 같으면 빠져나온다.
      // void가 있지만 return의 값이 없으므로 사용
    }
    System.out.println("나의 별명은" + nick + "입니다." );
  }

  public static void main(String[] args) {
    c3_other_ruturn sample = new c3_other_ruturn();
    sample.sayNick("홍길동"); // nick에 홍길동을 넣는다.

    sample.sayNick("fool"); // nick에 fool을 넣는다.
    // "fool"이 동일하므로 그냥 빠져나가서 아무일도 일어나지 않음

    sample.sayNick("이민호"); // nick에 이민호을 넣는다.

  }

}
출력값 :
나의 별명은 홍길동입니다.
나의 별명은 이민호입니다.
```

메서드 안에서의 변수

```
package day11_1014;
public class C4_Method_V {
  void varTest(int a) {
    a++;
      }
  public static void main(String[] args) {
    int a = 1;
    C4_Method_V sample = new C4_Method_V();
    sample.varTest(a);
    System.out.println(a);
  }
}

출력값 : 1

먼저 메인 메서드에서 a라는 변수를 1에 대입하여 변수를 생성했다.
그리고 varTest를 호출하면서 a를 던졌다.
```

그 다음에 a를 출력했더니 당연히 varTest 메서드에서 a의 값을 1 증가 시켰기 때문에
 2가 출력이 될 것으로 생각했으나 실행 결과는 1이 나왔다.

그 이유는 메서드에서 사용되는 변수는 메서드 안에서만 쓰여지는 변수이기 때문이다.
이렇게 메서드에서 쓰이는 변수는 메서드 밖의 변수이름과는 전혀 상관이 없다.
따라서 해당 메서드 안에서만 쓰이는 변수라는 의미로 로컬 변수(local variable)하고
한다.

진짜로 1을 증가 시키고 싶을 때는

- return을 이용한 결과 값을 가지고 나가는 방법

```
package day11_1014;
public class C4_Method_V {
  int varTest(int a) {
    a++;
    return a; // 또는 return ++a;
  }
  public static void main(String[] args) {
    int a = 1;
    C4_Method_V sample = new C4_Method_V();
    a = sample.varTest(a);
    System.out.println(a);

  }
}

// a++ 과 ++a 차이
// a++는 a가 먼저 나가고 1을 더함
// ++a는 값에 1을 더하고 a가 나감
```

또 다른 방법
- 전역 변수(global variable)를 이용
- 좋아보이지만 전역변수보다 지역변수를 사용하자

```
package day11_1014;
public class C4_Method_V {

  int a;

  void varTest(C4_Method_V sample) {
    methodv.a++;
  }
  public static void main(String[] args) {
    //int a = 1;
    C4_Method_V sample = new C4_Method_V();

    sample.a = 1 ;
    sample.varTest(sample);
    System.out.println(sample.a);

  }
}
```

위에서는 a라는 int 자료형 변수를 main매서드에서 선언했었는데
 여기서는 클래스의 객체변수로 선언했다.

그리고 varTest매서드는 클래스의 객체를 입력받아서 해당 객체의 객체변수 a의 값을
 1 증가 시키는 역할을 하도록 수정했다.

그리고 메인 매서드에서는 varTest매서드에 1이라는 값을 전달 하던것을 클래스 객체인
 sample을 넘기도록 수정했다.

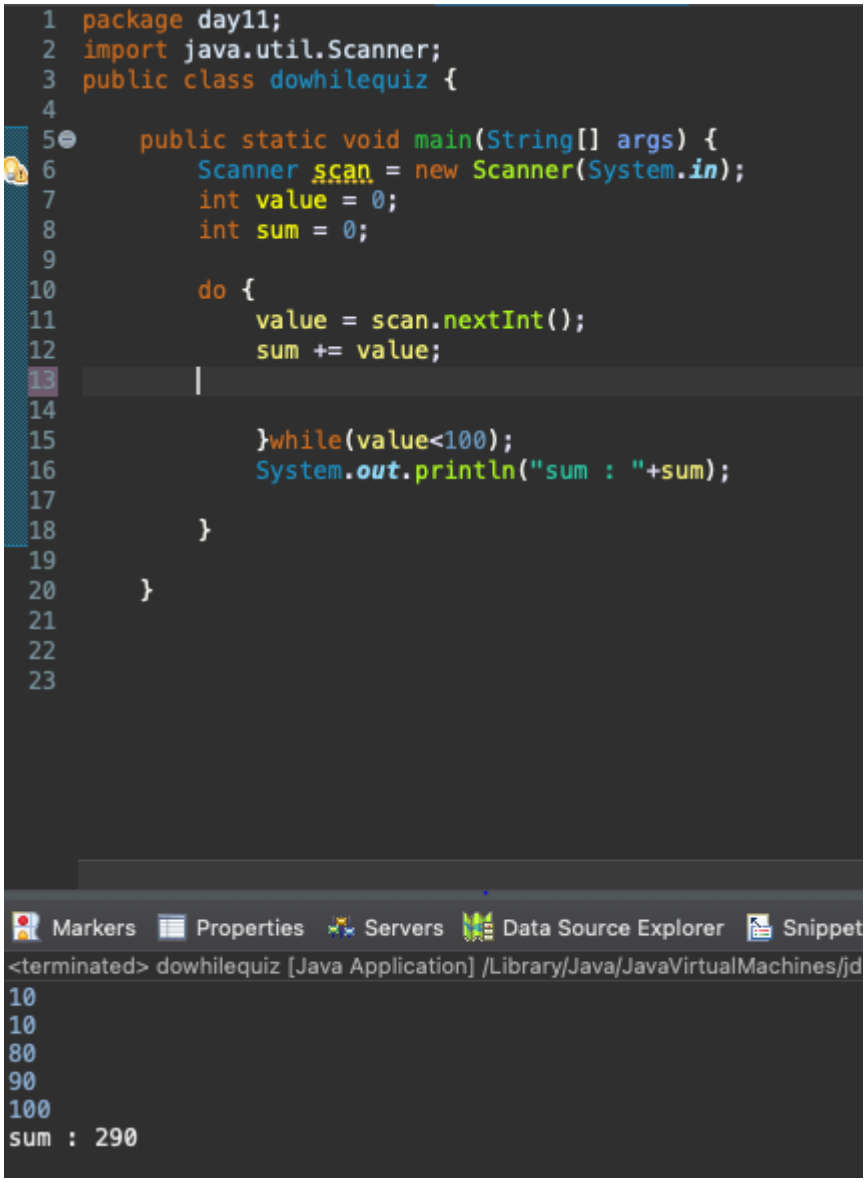
이렇게 수정을 마치고 실행하면 원래는 1이었는데 원하는대로 1이 증가된 2가 최종
출력되는 것을 확인할수 있다.

삼항연산자

여러개일때

```
1  package day11;
2
3  public class quiz5 {
4
5      public static void main(String[] args) {
6          // TODO Auto-generated method stub
7          int a = 4;
8
9          String name = (a == 1) ? "월요일"
10                 : (a == 2) ? "화요일"
11                 : (a == 3) ? "수요일"
12                 : (a==4)? "목요일"
13                 : (a == 5) ? "금요일"
14                 : ( a == 6) ? "토요일" : "일요일";
15
16
17         System.out.println(name);
18     }
19
20 }
21
```

Do While 예제


package day11;
import java.util.Scanner;
public class dowhilequiz {
public static void main(String[] args) {
Scanner scan = new Scanner(System.in);
int value = 0;
int sum = 0;
do {
value = scan.nextInt();
sum += value;
}while(value<100);
System.out.println("sum : "+sum);
}
}
Markers
Properties
Servers
Data Source Explorer
Snippet
<terminated> dowhilequiz [Java Application] /Library/Java/JavaVirtualMachines/jd
10
10
80
90
100
sum : 290