1주차 정리 [HOMEWORK-4a]

Date	@2023년 3월 24일
# percent	100
Tags	과제 국비
Status	Done
비고	

▼ variable

▼ 변수를 만드는 법

- 데이터 타입을 적는다 (int, float, double, long 등)
  int: 정수형
  long: 더 큰수를 표현할 수 있는 정수형
  float: 부동소수점
  double: 더 정밀한 부동소수점 (다만 실제 작업하면서 double type 을 사용하는 경우는 적다. -느림)
- 변수 이름이 필요하다면 작성한다.
- 필요하다면 초기화를 진행한다.

> 결론적으로 변수를 만들 때 가장 중요한 것은
> 변수 이름인데 명시성을 통해 함께 작업하는 팀원들에게
> 이것이 무엇을 의미하는 것인지 명확하게 전달하기 위한 목적이 가장 중요.

- 문자열 + 숫자의 경우엔 앞에 문자열이 나왔기 때문에
  자동으로 숫자를 문자열 처리하여 화면에 처리합니다. toString()

▼ final

- final을 사용하는 이유: 우선 상수로 고정 시킬 수 있다는 이점이 있음
- 상수로 고정 시킬 수 있다는 것의 이점:
  - 아래 있는 코드를 보면 3.3f를 직접 기입하고 있으므로 향후 프로그램이 커지면 직접 3.3f를 직접 기입하고 있으므로 향후 프로그램이 커지면 직접 3.3f를 작성한 부분을 모두 찾아서 값 변경에 따라 모든 코드를 수정해야 하는 불편함이 발생. 반면 TAX상수에 숫자를 기입하고 이 상수를 사용한다면 변동 상황이 발생할 때 해당하는 TAX수치값만 한번 변경하면 모든 작업이 일괄처리 됨.

```
final float FULL_PERCENT = 100;
final float TAX = 3.3f;
int income = 1000000;

System.out.println("프리랜서 세전 수입: " + income + ", 세후: " + (int)(income * (FULL_PERCENT - TAX)/FULL_PERCENT))
```

  - 불변 객체 (Immutable Object)
    클래스를 인스턴스화 하여 객체를 만들고 이게 불변이라면?

    final 변수는 새로운 값을 대입하거나 덧셈, 뺄셈등등이 불가능.
    결론적으로 입력되는 값을 변경하지 못하게 막음으로서
    원래 동작해야하는 동작의 무결성을 보장하게됨.

▼ flowControl

▼ for loop

for 문을 만드는 방법

1. for를 작성하고 소괄호{}를 작성후 중괄호[]를 작성한다.
2. 소괄호 내부는 다음과 같이 구성된다.
   (초기화; 조건; 증감)

```
for (statement1; statement2; statement3) {
//Statement 1 is executed (one time) before the execution of the code block.
//Statement 2 defines the condition for executing the code block.
//Statement 3 is executed (every time) after the code block has been executed.
}
```

- 여기서 초기화란 for문을 최초로 만나는 순간에만 동작한다. (그러므로 필요하다면 없어도 된다)
- 조건은 while, if, switch등에서 봤던 조건식과 동일.
- 조건이 만족하는 동안 for문이 반복됨.
- 증감또한 없어도 됨.
  표현을 조금 더 예쁘게 만들어 주기 위해 증감파트가 존재한다고 봐도 무방.

3. 중괄호 내에는 for 문을 반복하여 작업할 내용을 적어줍니다.

> 중요한 것은 어찌되었든 for 문은 조건 파트가 참인동안은 언제든지 반복된다.
> 초기화나 증감파트는 곁다리일뿐.

▼ if

- if문을 만드는 법
  - if 를 작성하고 소괄호() 를 작성하고 중괄호{}를 작성합니다.
  - 소괄호 내부에는 조건식을 작성합니다.
  - 중괄호 내부에는 조건이 만족된 경우 동작할 코드를 작성합니다.

> 만약 if, else if, else if, else if 형태로 코드가 작성된다면 조건식을 첫 번째 if가 만족되지 않았을 때 else if 를 보게 되므로 기본적으로 해당 else if에서는 if의 조건 또한 만족하지 않음을 내포하게 됩니다. 그리고 그 다음 else if 에서는 맨 처음 if가 만족하지 않고, 그 다음 else if를 만족하지 않고, 그리고 현재의 else if 조건을 만족해야 합니다.
> 그러므로 depth가 깊어질수록 코드를 파악하기 위한 혼동이 가중된다는 문제가 있습니다.
> 이와 같은 이유때문에 코드를 작성할 때 if, else if, else if보다는
> 그냥 if, if, if 형태가 더 좋습니다 (전제조건: 서비스 개발자)

▼ switch

- switch문을 작성하는 방법
  1. switch를 적고 소괄호()를 작성하고 중괄호{}를 작성합니다.
  2. 소괄호 내부에 switch case에서 사용할 조건을 적습니다.
  3. 중괄호 내부에는 case조건들을 적고
     각 조건에 대응하는 코드를 작성하면 됩니다.

```
switch(expression) {
  case x:
    statement;
    break;
  case y:
    statement;
    break;
  default:
    statement;
}
```

- Enhanced Switch Expressions (Java 14)
  - 특징으로는 가독성을 상승시키고 , break 누락으로 인한 오류가능성을 줄인다
  - 화살표 case label, 다중 case label, switch 연산식, yield예약어등의 특징이 있다.

```
//1. 화살표 case label
switch(expression) {
  case x -> statement;
  case y -> statement;
}

//2. 다중 case label
switch(expression) {
  case x, y -> statement;
  case a, b -> statement;
}

//3. switch연산식
int num = switch(expression) {
  case x -> intNum1;
  case y -> intNum2;
};

//4. yield 예약어
int num = switch(expression) {
 case x -> {
    statement;
    yield intNum1;
  }
  case y -> {
    statement;
    yield intNum2;
  }
};
```

▼ WhileLoop

While문 작성방법

```
int idx = 0;
final char ch = 'A';
  while (idx < 10) {
  System.out.println("idx: " + idx + ", 안녕: "+ (char)(ch + idx));
  idx++;
}
```

1. while을 작성하고 소괄호()를 작성하고 {}중괄호 작성
2. 소괄호 내부에 조건식을 작성
3. 중괄호 내부에 조건이 만족되는 동안 반복시킬 코드를 작성합니다.

▼ array

▼ stack array

```
final int START = 0;
final int[] numberArray = {1, 2, 3, 4, 5};
for (int i=START; i< numberArray.length; i++) {
  System.out.println("배열 출력: " + numberArray[i]);
}

//for each 응용
System.out.println();
  for (int num: numberArray) {
      System.out.println("배열 원소 출력: " + num);
  }
```

배열 만드는 법

1. 데이터 타입을 적고 대괄호 작성.
2. 변수 선언하듯 변수 이름을 작성
3. 필요하다면 중괄호를 열고 초기화 하거나, 또는 new datatype[] 형태로 heap메모리에 메모리 할당을 강제할 수 있음.

> Heap메모리와 대조되는 것이 Stack메모리이다.
> 위의 있는 코드는 Stack이라는 지역 변수에 설정하는 배열이며
> new 를 통해 할당하는 경우엔 Heap이라는 공간에 할당한다.
> 단, 우리는 로우 시스템 개발자가 아니므로 여기서 더 깊게 들어갈 필욘 없음.
> 그냥 new를 했기 때문에 Heap에 있구나 정보면 OK!
>
> 다만 궁극적으로 Stack 과 Heap의 차이가 무엇인가?
> Stack은 루프에서
> loop {
> final int data;
> }
> 형태로 있다면 data는 루프마다 초기화 되는 것을 확인 할 수 있다.
> 이런 지역변수 특성을 갖는 녀석들은 다 Stack이다.
>
> 반면 new를 해서 Heap에 설정되는 정보들은 메모리에 상주하게 된다.
> 그러므로 언제 어디서든 데이터에 접근할 수 있게 된다.
> 자바 개발자에게 있어 둘의 차이점이라면
> 현재 이 내용이 가장 크다고 볼 수 있음.
>
> 결론.
> Stack은 {}중괄호 내에서 사용됨.
> Heap은 new하고 이후 사용됨.

- for의 변형 버젼 for-each
  1. 배열의 데이터 타입을 작성한다.
  2. 배열의원소를 표현할 이름을 적당히 지정한다.
  3. 콜론을 하나 찍는다.
  4. 정보를 하나씩 꺼내올 배열을 적는다.

▼ Heap array

```
final int START = 0;

final int ALLOC_ARRAY_NUMBER = 5;
final int[] numberArray = new int[ALLOC_ARRAY_NUMBER];

for (int i=START; i < ALLOC_ARRAY_NUMBER; i++) {
    numberArray[i] = i + 1;
    System.out.printf("numberArray[%d] = %d\n", i, numberArray[i]);

System.out.println();
for (final int num: numberArray) {
    System.out.println("numberArray elem: " + num);
}
```

- numberArray를 final로 선언한 것은 Heap에 할당된 메모리 변경 금지를 요청한 것이고 내부에 배치하는것에는 영향을 받지 않음.
- final 선언은 새로운 메모리를 할당해서 전달하는 것을 막는다.
  조금 풀어보자면 객체를 상수화 하느냐
  객체 내부의 값을 상수화 하느냐의 관점으로 봐야한다.
  현재 관점은 객체를 상수화 하였기 때문에 다른 객체를 대입한다면 금지된다.

▼ Syste.out.printf()

printf 의 경우 format을 출력한다는 뜻으로 printf이다.
format은 %d의 경우 정수형(int)
%s의 경우 String,
%f의 경우에는 (float, double)을 처리한다.

```
System.out.printf("numberArray[%d] = %d\n", i, numberArray[i])
```

첫번째 %d는 i 가 대응해서 대체되고
두번째 %d는 numberArray[i]가 대응해서 대체된다.

▼ class

```
class Led{
  private Boolean isTurnOn;
  public Led() {
    this.isTruenOn = false;
  }

  public boolean getTurnOn() {
    return isTurnOn;
  }

  public void setTurnOn(Boolean is TurnOn) {
    this.isTurnOn = isTurnOn;
  }
```

> Led클래스는 불이 켜졌다 혹은 꺼졌다만 관리하면 됨.
> 클래스의 형태는 다음과 같다.
>
> ——Led 객체——
> | isTurnOn |
> ——Led 생성자—
> | Led 생성자 |
> | getTurnOn |

> | setTurnOn |
> ————————————-

- 생성자는 class의 이름과 동일하며 리턴타입이 없다.
- 만드는 규칙
  1. public을 적고 class이름을 적은후 소괄호와 대괄호 작성.
  2. 만약 외부에서 값을 입력받을 것이라면 소괄호에 입력받을 형태를 작성
  3. 실제 클래스가 new를 통해 객체화 될때 구동시키고 싶은 작업을 중괄호 내부에 배치
- 클래스 내부에 기능을 수행하는 집합들을 메서드라고 한다.
- public 메서드를 작성하는 법
  1. public 을 작성하고 소괄호, 대괄호 작성.
  2. 리턴타입을 public 옆에 작성
  3. 메서드의 이름을 그 옆에 작성
     (이 메서드의 기능을 명시적으로 나타내주는 이름으로 선택하여 작성)
  4. 중괄호 내에서는 실제 메서드 이름에 해당하는 작업을 진행.
     이 때 단순히 작업만 하고 정보 반환이 없다면 리턴 타입은 void이다.
     참/거짓이라면 Boolean이며, 정수라면 Long또는 Integer이다.
     무엇을 리턴 하느냐에 따라 적절한 형태를 적는다.

> DDD
> 사실 Led 예시의 경우 Led 스스로 켤수 없기 때문에 다른 객체의 도움을 받아야 함.
> 수십년간의 SW전문가들이 이러한 개념들을 정리하였고
> 그 개념의 일환으로 탄생하게 된 개념이 Domain Service 개념입니다.
> Domain Service를 만들어서 얻는 이점은
> 비즈니스 관점을 좀 더 명확하게 만들어 준다는 이점이 있습니다.
> 실제로 전구를 켜고 끄는 작업을 가지고 비즈니스를 할 만한 것들이 많지는 않지만
> 그래도 Domain Service를 나눠본다면 아래와 같은 것들이 존재할 것입니다.
> 켜기, 끄기, 깜빡이기
>
> → 조금 쉽게 접근해 보자면 Domain Service는
> 객체 스스로가 직접 하기에는 표현이 애매해지는 작업들을
> 모두 Domain Service로 재배치 하게 됩니다.
> 이를 통해 가독성과 유지 보수성의 향상을 가져올 수 있습니다.
>
> 제어하는 Controller 에 RequestForm 개체를 전달
> RequeastForm 객체는 Domain Service에서 처리하기 적합한 형태로 Request로 변환
> 그리고 Request를 보고 적절한 Entity를 추출하게 되는데
> 이런 관점으로 접근하면 setter를 완벽하게 제거 할 수 있습니다.
> 즉, 필요하면 final 객체에 final 변수들을 설정해서 전달한다는 의미입니다.

> OOP (Object Oriented Programing)
> DDD (Domain Driven Development)

> 잘 만든 OOP란 무엇인가?
> OOP란 모든 정보를 객체화 하여 레고처럼 필요하면 조립하여 관리하자라는 뜻을 갖고 있습니다.
>
> 하지만 모든 정보를 하나의 클래스에서 관리하게 되는 경우
> 객체가 비대해지면서 해당 객체가 어떤 목적을 가지고 있었는지 목적성을 잃게 됩니다.
> Domain이라는 관점은 이렇게 클래스가 어떤 주제에 집중을 하고 있는지를 본다고 생각하면 됩니다.
> 즉 내가 집중하는 주제가 무엇인가를 알 수 있도록 예쁘게 잘 표현해주는 것을 OOP라 봐도 무방합니다.

▼ math

랜덤을 만드는 방법

1. Math.utility.random()을 작성한다.
2. 최소값과 최댓값을 확인한다.
3. 최소값은 더하기로 표기한다.
4. 최대값은 곱하기로 표기한다
   (실제 최댓값 계산은 곱하는 값 + 최소값 - 1)

```
int randomNumber = (int) (Math.random() * (MAX -MIN + 1)) +MIN;
```