

5주차 과제

study 할래?

스터디 내용

1. 클래스 정의하는 방법
2. 객체 만드는 방법 (`new` 키워드 이해하기)
3. 메소드 정의하는 방법
4. 생성자 정의하는 방법
5. `this` 키워드 이해하기
6. (Optional)과제

- `int` 값을 가지고 있는 이진 트리를 나타내는 `Node`라는 클래스를 정의
- `int value`, `Node left`, `right`를 가지고 있어야 합니다.

- `BinaryTree`라는 클래스를 정의하고 주어진 노드를 기준으로 출력하는 `bfs(Node node)`와 `dfs(Node node)` 메소드를 구현하세요

- DFS는 왼쪽, 루트, 오른쪽 순으로 순회하세요.

1. 클래스를 정의하는 방법

클래스란!?

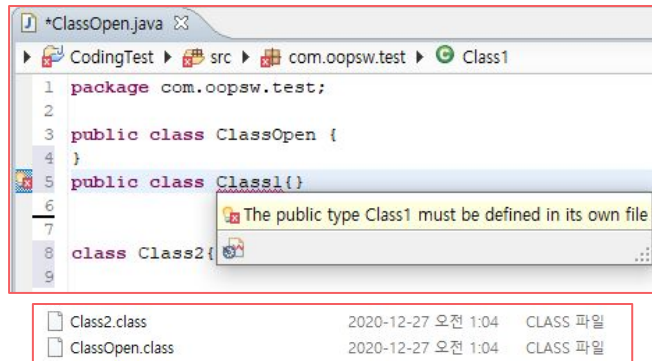
- 객체들의 협력 관계를 코드로 옮기는 도구이다.
- 협력에 참여하여 메시지를 주고 받는 객체를 만드는데 필요한 구현 메커니즘

클래스를 정의하는 방법

클래스 이름은 다른 클래스와 식별할 목적으로 사용되며 자바의 식별자 규칙에 따라 만든다



1. 클래스 선언하는 방법



일반적으로 클래스이름.java 파일 하나당 하나의 클래스를 선언하지만 두개 이상도 가능하다. 두개 이상의 클래스를 가진 파일은 컴파일하면 바이트 코드 파일은 클래스를 선언한 개수만큼 생긴다. 파일 이름과 동일한 이름의 클래스에만 **public** 접근 제한자를 붙일 수 있으며 가급적 하나당 클래스 하나를 선언하는것이 좋다.

```
public class ClassOpen {
    //field
    int data;

    //constructor
    ClassOpen() {}

    //method
    void method() {}
}
```

클래스에는 객체가 가져야 할 구성 멤버가 선언된다.

- 필드(Field) : 객체의 데이터가 저장되는 곳
- 생성자(Constructor) : 객체 생성 시 초기화 역할
- 메소드(Method) : 동작에 해당하는 실행 블록

Constructor 선언 생략하면 컴파일러가 **default constructor** 생성
new 연산자로 호출되는 {}블록
No return type

2. 객체 만드는 방법

프로그래밍에서 객체지향이란?

프로그래밍 패러다임의 한 종류로서 프로그래밍 패러다임에는 대표적으로 C언어(절차형 패러다임), 자바(객체지향 패러다임), 리스프(함수형 패러다임), 프롤로그(논리형 패러다임)이 있다. 각 패러다임에는 문제를 해결하는데 필요한 개념을 지원한다.

“

프로그래밍 패러다임은 개발자 공동체가 동일한 프로그래밍 스타일과 모델을 공유할 수 있게 함으로써 불필요한 부분에 대한 의견 충돌을 방지하고 프로그래밍 패러다임을 교육시킴으로써 동일한 규칙과 방법을 공유하는 개발자로 성장 할 수 있도록 준비시킬 수 있다.

오브젝트 - 조영호님 프로그래밍 패러다임 중

”

2. 객체 만드는 방법

프로그래밍 패러다임은 공동체가 동일한 규칙과 표준에 따라 효율적으로 프로그램을 작성할 수 있게 하기 위한 프로그래밍 스타일 가이드이고, 자바는 객체지향 패러다임 기반의 언어이다.

“

객체지향이란 시스템을 상호작용하는 자율적인 객체들의 공동체로
바라보고 객체를 이용해 시스템을 분할하는 방법이다.

자율적인 객체란 상태와 행위를 함께 지니며 스스로 자기 자신을 책임지는 객체를 의미한다.

객체는 시스템의 행위를 구현하기 위해 다른 객체와 협력한다.
각 객체는 협력 내에서 정해진 역할을 수행하며 역할은 관련된 책임의 집합이다.

객체는 다른 객체와 협력하기 위해 메시지를 전송하고,
메시지를 수신한 객체는 메시지를 처리하는 데
적합한 메서드를 자율적으로 선택한다.

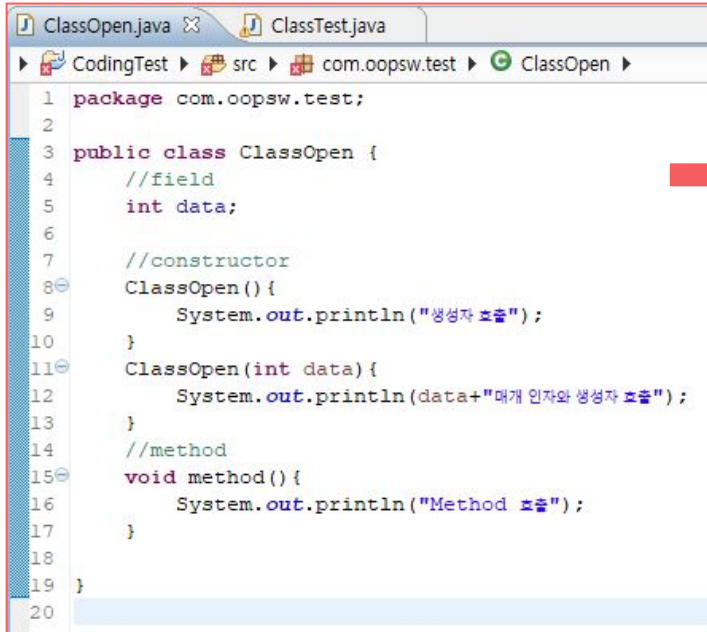
객체지향의 사실과 오해 - 조영호님 객체지향의 본질 중

”

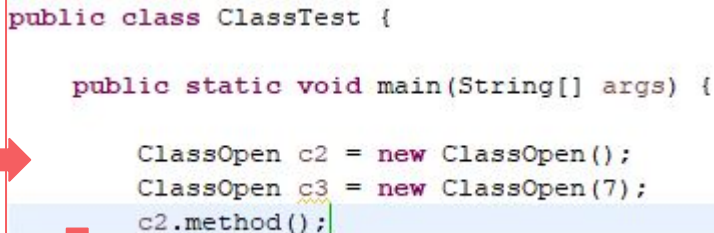
2. 객체 만드는 방법(new 키워드 이해하기)

```
new ClassOpen(); //1  
ClassOpen c1; //2 선언  
c1 = new ClassOpen(); //2-1 초기화  
ClassOpen c2 = new ClassOpen(); //3 동시에
```

객체 생성은 new 연산자를 사용하면 된다.



```
1 package com.oopsw.test;  
2  
3 public class ClassOpen {  
4     //field  
5     int data;  
6  
7     //constructor  
8     ClassOpen() {  
9         System.out.println("생성자 호출");  
10    }  
11    ClassOpen(int data) {  
12        System.out.println(data+"대개 인자와 생성자 호출");  
13    }  
14    //method  
15    void method() {  
16        System.out.println("Method 호출");  
17    }  
18  
19 }  
20
```



```
public class ClassTest {  
  
    public static void main(String[] args) {  
  
        ClassOpen c2 = new ClassOpen();  
        ClassOpen c3 = new ClassOpen(7);  
        c2.method();  
    }  
}
```

생성자 호출
7대개 인자와 생성자 호출
Method 호출

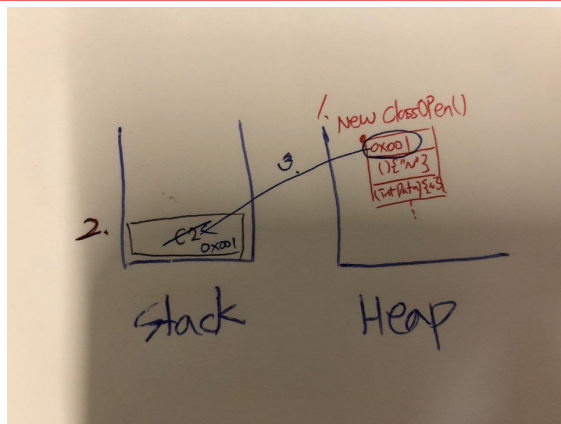
new로 생성된 객체는 Heap에 생성되고 Heap에서 객체의 주소를 리턴하면 그 주소를 클래스 변수에 저장한 뒤 객체를 사용한다. 이 과정을 인스턴스화 라고 하며, 이렇게 만들어진 c2,c3객체를 클래스의 인스턴스라고 한다.

2. 객체 만드는 방법(new 키워드 이해하기)

```
public class ClassTest {  
  
    public static void main(String[] args) {  
        ClassOpen c2 = new ClassOpen();  
        ClassOpen c3 = new ClassOpen(7);  
        c2.method();  
    }  
}
```

new 키워드의 경우

문제해결을 위해 타객체의 도움을 받고 싶을 때
새 선수를 임대 하는 것처럼 new로 연결을
시켜주는 것으로 이해하면 될 것 같다.



+메모리간의 데이터 전달은 주소값으로 한다

3. 메소드 정의하는 방법

메소드는 선언부와 실행블록으로 정의한다.

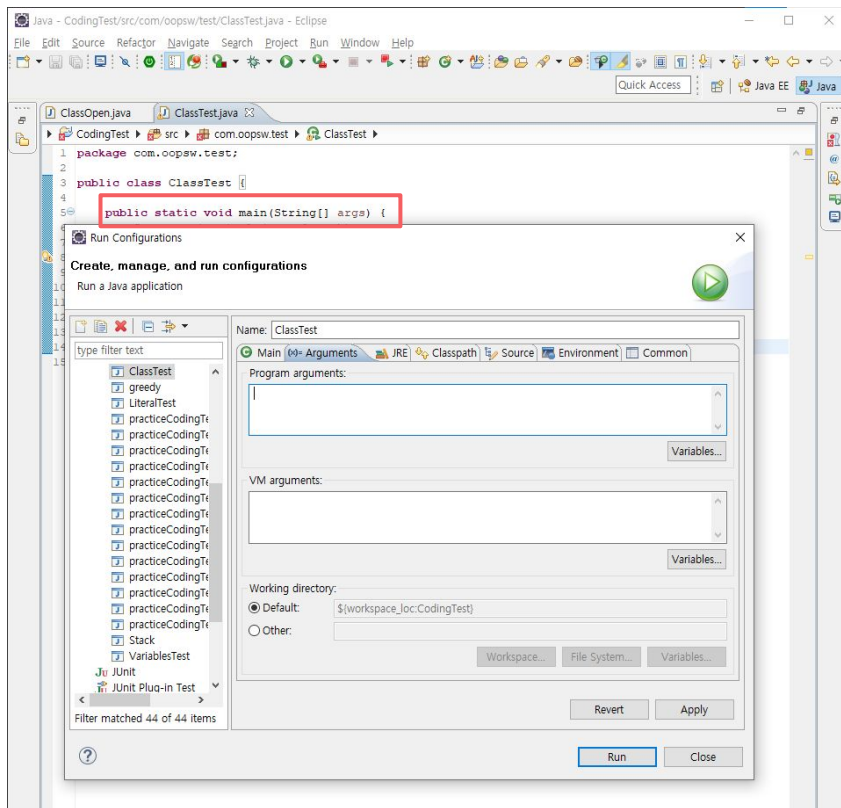
```
//method  
void method(){  
    System.out.println("Method 호출");  
}
```

사진과 같이 리턴타입 메소드이름(매개인자){실행}으로 정의되며 리턴타입에는 기본타입과 사용자 정의 타입이 존재한다.

동시에 메소드는 오버로딩이 가능하다. **Overloading** 매개인자가 올 경우들을 미리 정의해 놓는다. 같은 이름의 메소드에 다양한 매개변수!

리턴타입과 매개변수 이름은 오버로딩에 영향받지 않는다.

3. 메소드 정의하는 방법



`public static void main(String[] args){`

} 자주쓰는 이것또한 메서드이다.

main은 자바컴파일러가 실행되면 가장 먼저 기본으로 실행하는 메서드이며

매개인자의 **String[] args**는 run → run

configuraions에서 프로그램 실행시 매개인자를 입력할 수 있다.

물론 기본은 **String[]** 타입이며 변경가능하다.

4. 생성자 정의하는 법

`new` 연산자로 호출되는 특별한 중괄호 블록이며 생성자의 역할은 객체 생성 시 필드를 초기화하고 메소드를 호출해서 객체를 사용할 준비를 한다.
생성자는 메소드와 비슷하게 생겼지만 클래스 이름으로 되어있고 리턴 타입이 없다.

```
public class ClassOpen {  
    //field  
    int data;  
  
    //constructor  
    ClassOpen() {  
        System.out.println("생성자 호출");  
    }  
    ClassOpen(int data) {  
        System.out.println(data+"매개 인자와 생성자 호출");  
    }  
}
```

5. this 키워드 이해하기

- **this**는 객체 자신을 의미한다. 인스턴스의 주소가 저장되어 있다.
- 모든 인스턴스 메소드에 지역변수로 숨겨진 채로 존재한다.
- 객체 내부에서 필드에 접근하기 위해 **this**를 사용할 수 있다.
- 주로 생성자와 메소드의 매개변수 이름이 필드와 동일한 경우 필드를 명시하기 위해 사용한다
- 한 생성자에서 다른 생성자를 호출한다. 단, 반드시 첫 줄에서만 호출이 가능하다.
- 생성자 내에서 다른 생성자를 호출할 때 클래스 이름 대신 **this**를 사용해야 한다.

6. Optional