12.20 火 _ 객체지향 언어 1

❗ `객체지향 언어`는 기존의 프로그래밍 언어에 몇 가지 새로운 규칙을 추가해 보다 발전된 형태의 것

이러한 규칙들을 이용해서 코드 간에 서로 관계를 맺어 줌으로써 보다 **유기적으로 프로그램을 구성**하는 것이 가능

※ `프로그램`은 기능적으로 설계 하기❗ ※
※ `객체지향 언어`는 거시적 관점에서 설계 하기❗ ※

프로그래밍 언어의 발전史❗

비구조적 언어 (통짜로 짠다)		구조적 언어 (배운 것)	⇒	객체지향적 언어 (Object Oriented Programming Language)
Machine language(기계어)	+ 재활용	Flow control (선택 , 반복)	+ 재사용성增加 →	→ **상속** (Inheritance) 으로 **재활용률**을 增加！ , **상속**이 **OOP**의 **핵심**이다.
CPU 가 알아 먹는 언어	+ High-level	*Function* (기능별로 나눠 모듈화)	+ `Module` 化 (유지 보수가 용이)	**金 때문에 발전이 되었다** 기능에 따른 객체를 중심으로 자료의 추상화, 상속, 다중상속, 다형성, 동적바인딩의 특징을 가짐 , 기능에 따른 독립적인 단위 즉 객체를 중심으로 프로그래밍
one code → 한번에 다 작성	+ `Compiler` 도입			`객체지향언어`의 가장 큰 장점 : 코드의 재사용성이 높고 , 유지보수가 용이하다
수정하기 어렵다				⇒ 프로그램 개발과 유지보수에 드는 시간과 비용을 획기적으로 개선
조건식 == true , Goto 文				**"클래스" 단위로 OPP 를 작성 한다**

❓ 왜 이렇게 프로그래밍 언어가 개발되었는가? / oop를 왜 사용 하나?
⇒ **돈** 때문이다 (개발 비용[**인건비**] 때문이다)
⇒ 재활용이 중요한 이유는 개발 시간을 줄이기 위함이다 , 시간이 줄면 그 시간에 다른 프로젝트를 할 수 있다 = 돈

< 구조적 언어 구성 요소 5가지 >

Data ⇒	**Comment**	⇒ Information
	Variable → Array (List)	
	Operator	
	Flow control (if ,while , for)	
	Function	

⇒

OOP!!

1. Class and Object
2. **Inheritance (상속) ⇒ 현존하는 세계에서의 "상속"의 意味와 동일하다**
3. Modifer (제어자)
4. Polymorphism (다형성)
5. Abstract class (추상화 , 추상 클래스)
6. Interface (인터페이스)
7. Exception handling (예외처리)

⇒ 4,5,6 번은 **상속**이 없으면 돌아가지 않는다

클래스(class) 와 객체

❓ 클래스(*class*)

클래스의 정의 - **객체**의 모든 속성과 기능을 **정의**해 놓은 것 (객체의 설계도 • 틀)
클래스의 용도 - **객체를 생성**하는데 사용

⇒ 클래스로부터 객체를 생성하면, 클래스에 정의된 속성과 기능을 가진 객체가 만들어 지는 것
⇒ 클래스는 객체(object)를 여러 개 만들려고 사용한다. 클래스 = 객체를 찍어내기 위한 틀
⇒ 클래스만 선언하면 메모리 상에 안 올라간다 (코드는 올라간다) / 아무것도 안 일어난다

클래스 구조

> 클래스는 class 키워드와 함께 클래스명을 표기 , 클래스명의 첫 글자는 반드시 대문자로 시작!

```
Class Betty {
  .........
}
```

❓ 객체

실제로 존재하는 것 ,사물 또는 개념
프로그래밍에서 **객체**는 클래스에 정의된 내용대로 메모리에 생성된 것을 뜻함
⇒ ※ 클래스만 선언하면 메모리에 안 올라가고 아무것도 안 일어난다 ※

객체는 속성과 기능 두 종류의 구성 요소로 이루어져 있고 , 다수의 속성과 다수의 기능을 갖는다.
⇒ ⭐⭐⭐ 객체는 속성과 기능의 집합 ⭐⭐⭐

⇒ ⭐ 객체를 사용한다는 것은 객체가 가지고 있는 속성과 기능을 사용한다는 뜻 ⭐

※ 클래스는 단지 객체를 생성하는데 사용될 뿐, 객체 그 자체는 아니다 ※

※ 프로그래밍에서는 먼저 클래스를 작성한 다음, 클래스로 부터 객체를 생성하여 사용한다 ※

※ 객체를 상속하기 위해 프로그램의 작성 단위 Object 라는 단위를 쓴다 / 상속 시키는 단위를 클래스 등으로 사용한다 ※

method ⇒ 조직적 절차, 방법 , 순서, 절차, 체계 , 체계성

객체의 구성요소 - 속성과 기능

- 객체가 가지고 있는 속성과 기능을 그 객체의 `멤버(member)` 라 한다
- 객체지향 프로그래밍에서는 **속성**과 **기능**을 각각 `변수` 와 `메서드=함수` 로 표현한다

속성 (property) [`변수` **]** —> **멤버 변수 (variable) [** 클래스 안의 **변수]**

기능 (function) [`함수` **]** —> **멤버 메서드 (method) [** 클래스 안의 **함수]**

객체와 인스턴스 (사실상 같은 의미)

❓ 인스턴스 **(instance)**

어떤 클래스로 부터 만들어진 `객체` 를 그 클래스의 `인스턴스 (instance)` 라고 한다

※ 클래스에서 객체를 만드는 과정은 생성자가 수행한다.
`인스턴스화 (instantiate)` : 클래스의 생성자로 부터 객체를 만드는 과정

※ 인스턴스와 객체는 같은 의미이므로 두 용어의 사용을 구분할 필요는 없지만 ,문맥에 따라 구별하는 것이 좋다

★★★한 파일에 여러 class 작성 ★★★

❗ public class 가 있는 경우 , **소스 파일의 이름은 반드시 public class의 이름과 일치**해야 한다!! and 둘 이상의 public class 가 존재 불가 !
소스 파일(*. java)과 달리 클래스 파일 (*. class) 은 클래스 마다 하나 씩 만들어진다

객체 생성 방법

`new` 연산자 ★

: 객체를 생성 한다 ⇒ 메모리 상에 올린다는 것

객체가 만들어진다는 것은 **선언한 멤버들**이 **메모리 상으로 올라간다**는 것 (메모리 상으로 올라갈 때 초기화를 할 수도 있다)

`new` 객체를 찍어낸다는 의미
⇒ `new` 우측에 찍고자 하는 `class` 이름 붙이기 → `new` 연산자를 사용하면 **메모리에 올라간다**. 그리고 **( )** 를 붙인다

`생성자` ★

메모리에 올라가면서 현재 만들어진 **객체 안에 값**들을 **초기화** 시키는 **알고리즘**이 들어있다.

해당 `class` 를 가지고 객체를 만들어 낼 때 `생성자` 는 **딱 1번만 실행** 된다

⇒ `new` 연산자를 이용해 해당 `class` 의 객체를 찍는다. 그 때 반드시 해당 `class` 의 `생성자` 를 호출해야 한다
`생성자` 가 하나도 없으면 자바에서 자동으로 `defort 생성자` 를 할당한다 / 한 개라도 `생성자` 가있으면 `defort 생성자` 는 할당 안 한다

참조 변수 ★

객체를 만들어 놓고 그 **객체에 접근** 할 수 있기 위해 참조 변수를 사용한다 (**주소 값**을 보고 찾아다)

> ! 변수를 선언하는데 변수의 자료형이 **class** 이름이면 참조 변수가 되고 , 해당 객체의 주소 값을 가지기 위해 사용된다

class / 클래스 안에 5가지가 들어올 수 있다 , 자바는 3가지

```
// Student Class
class Student {        // 프로그램에는 여러 class 가 있을 수 있기 때문에 이름을 붙여 구분한다

  // 1. 멤버 변수 (Member Variable)
  String stdName;       // 클래스안에 변수가 들어간다
  int subKorean;
  int subMath;
  int sum;

  // 2. 생성자 (Constructor)      // 형태는 메소드와 같으나 class의 이름과 같다
  Student(String argName, int argKorean, int argMath) {    // 반환하는 자료형이 없다 => 반환하는 값이 없다는 것
    stdName = argName;
    stdKorean = argKorean;
    stdMath = argMath;
    }

  // 3. 멤버 메소드 (Member Method)
  int getSum() {   // 여기 앞의 int는 이 메소드를 호출했을 때 반환하는 값의 자료형 = 반환형 !  => 이메소드를 호출하면 int 형이 나온다
    return subKorean + subMath;    // 메소드가 가지는 형태 =>  반환형 메소드 이름 ( ) { }
  }
}
```

```
// App
public static void main(String[] args) {
//  클래스명 참조변수명 = new 클래스명 ():
      Student std1 = new Student(); // err
      Student std2 = new Student(); // err
      Student std3 = new Student(); // err
// 호출하려 하는데 매개변수가 아무것도 없는 생성자가 없어서 error!!
     이렇게 작성해야 한다  =>  Student(String argName, int argKorean, int argMath)
// ★★★ 변수를 선언하는데 변수의 자료형이 class 이름이면 참조 변수가 되고 , 해당 객체의 주소 값을 가지기 위해 사용된다 ★★★
}
```

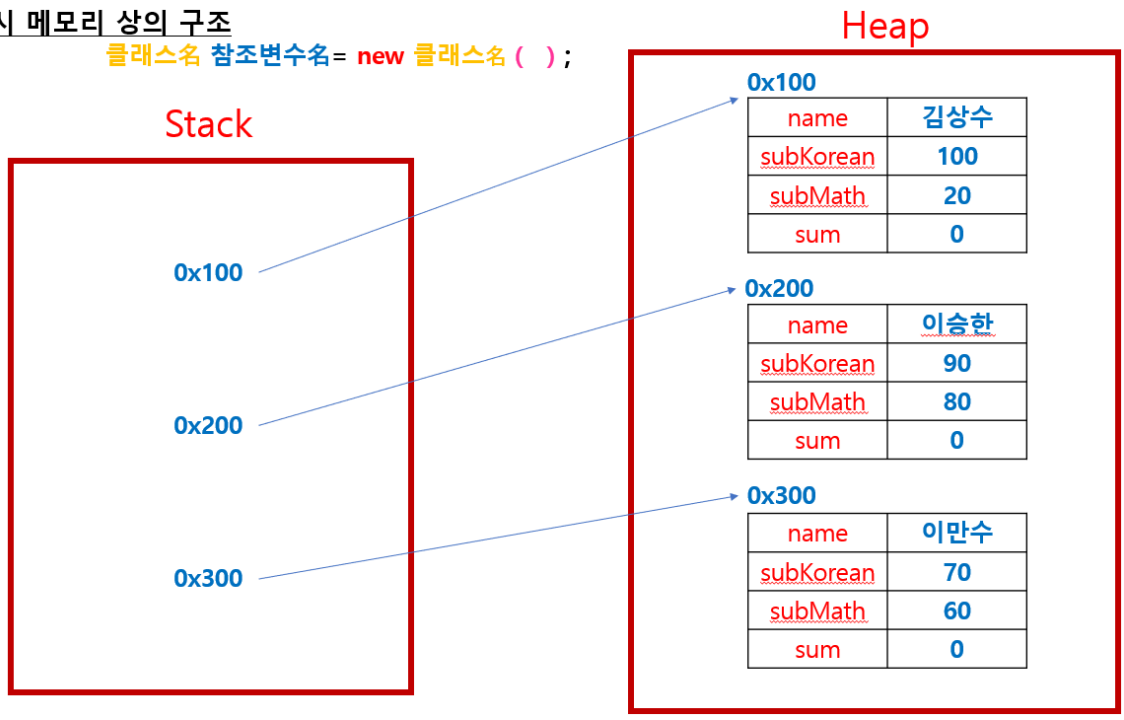
- 멤버 변수 : `subName`, `subKorean`, `subMath`...
- **생성자** : `Student(String argName, int argKorean, int argMath)`
- 멤버 메소드 : `getSum()`

올바르게 객체를 생성 하려면 이렇게!!

```
// App
public static void main(String[] args) {
  Student std1 = new Student("김상수", 100, 20);
  Student std2 = new Student("이승한", 90, 80);
  Student std3 = new Student("이만수", 70, 60);
}
```

실행 시 메모리 상의 구조

클래스名 참조변수名= new 클래스名 ( ) ;



❗ 하나의 `class`를 정의 해두면 여러 개의 객체를 찍어낼 수 있다!!
`class`를 만들면 동일한 멤버 변수, 동일한 메서도 ,동일한 생성자를 가지는 놈들을 여러 개 찍을 때 유용하게 사용할 수 있다

위의 코드 활용 예시

```
package Course;

import java.util.Scanner;

//  public class 명은 무조건 파일이름!!!
public class test1 {
    // 안에는 메인메소드가 들어가야한다 !! (반드시는 아님xxx)
    public static void main(String[] args) {

        Scanner scn = new Scanner(System.in);

        Student stdList[] = new Student[2];

        for (int i = 0; i < stdList.length; i++) {
            stdList[i] = new Student(scn.next(), scn.nextInt(), scn.nextInt());
        }

        for (int i = 0; i < stdList.length; i++)
            System.out.println(stdList[i]);
    }

}

// Student Class를 선언
class Student {

        // <<-- Member variable
        String stdName;
        int subKorean;
        int subMath;

        int sum; // -->>

          // Constructor : 생성자
          Student(String argName, int argKorean, int argMath) {
              stdName = argName;
              subKorean = argKorean;
```

```
            subMath = argMath;
        }

        // Member method
        int getSum() {
            return subKorean + subMath;
        }

        // 내부적으로 만들어져 있는 것을 고쳐 쓴다
        @Override
        public String toString() {
            // TODO Auto-generated method stub
            return stdName + "\t" + sum;
        }
}
```

+ 추가 내용 (책 참고)

객체는 사용할 수 있는 **실체**를 의미하며, **클래스**는 객체를 만들기 위한 **설계도**와 같다

따라서 1개의 정의된 **클래스**를 이용해 여러 개의 객체를 만들 수 있다.

❓ 클래스는 왜 사용할까?

- `변수` : 다양한 형태의 **데이터를 저장**하기 위해 각각의 데이터를 저장할 수 있는 변수라는 문법 요소를 만들어 사용했다
- `배열` : 데이터의 종류가 많아질수록 데이터의 개수만큼 변수명을 짓거나 관리하는 일이 버거워졌다. 이런 문제를 해결하기 위해 만든 문법 요소가 “배열” 이다.
  배열을 사용하면 **같은 자료형인 변수들을 묶어** 1개의 새로운 자료형으로 관리할 수 있으므로 **관리해야 할 변수의 개수를 현저하게 줄일 수 있다**.
- `구조체` : 배열은 같은 자료형만 묶을 수 있으므로 자료형이 다르면 1개의 배열로 관리할 수 없다.
  이를 보완하기 위해 만든 문법 요소가 “ 구조체 (Struct) “ 이다. 구조체를 사용하면 **서로 다른 자료형도 1개의 자료형으로 묶어 관리할 수 있다**.
- `클래스` : 이렇게 다양한 자료형의 데이터를 하나로 묶어 관리할 수 있는 구조체는 말 그대로 데이터만 묶어 놓은 것이다.
  여기에 반 평균 성적을 출력하거나 반 학생들의 총 점을 계산하는 등과 같은 **기능을 추가**하면 반의 성적과 관련된 **모든 내용을 효율적으로 처리**할 수 있다.
  이것이 바로 “클래스” 다. 즉 클래스는 **다양한 자료형의 데이터를 묶어 관리할 수 있을 뿐 아니라 데이터를 처리하는 다양한 기능까지 함께 관리하는 문법 요소**다.

> 클래스에 포함돼 클래스 안에 있는 데이터를 처리하는 기능을 “`메서드`”라고 한다.
> 각 객체에 포함된 데이터는 `속성` 또는 `필드` 라 하고, 기능은 `메서드`라고 한다

❗ 자바는 C++ 등과 같은 다른 프로그래밍 언어에서 `클래스`에 이르기까지 프로그래밍 문법 요소가 만들어진 이후에 개발됐기 때문에 `클래스`를 기본 문법 요소로 사용한다.

`절차지향형 프로그래밍(기능 중심 프로그래밍)` : 순서에 맞춰 단계적으로 실행하도록 명령어를 나열하는 방식

`객체지향형 프로그래밍(객체 중심 프로그래밍)` : 클래스를 사용한 후에, 객체지향형 프로그래밍 방식이 주로 사용되는데 , 프로그램을 객체 단위로 수행하는 방식

❗ `객체지향형 프로그래밍` 에서는 어떤 추가 사항이 있을 시 필요한 객체를 추가하고 기능을 호출하면 되지만 ,
`절차지향형 프로그래밍` 은 어떤 절차에 어떤 기능을 어떤 순서대로 넣어야 할지 難

자바에서 제공하는 객체지향 문법 요소

`객체지향 프로그래밍 언어` 인 `자바` 는 프로그램을 객체 단위로 구성해 상호 연동 시킴으로써 프로그램을 실행한다.

자바에서 제공하는 객체지향 문법 요소는 크게 2가지

`Class`

- `일반 Class`
- `추상 Class` : 1개 이상의 추상 메서드가 있는 Class

`interface`

: `추상 class` 의 특수한 형태 (모든 메서드가 추상 메서드) 라고 볼 수 있다.

클래스 구조

클래스는 class 키워드와 함께 클래스명을 표기 , 클래스명은 대문자로 시작!

```
Class 클래스이름 {
  .........
}
```

자바 소스 파일은 클래스의 외부 구성 요소와 내부 구성 요소로 나눠 생각할 수 있다.

betty 라는 이름의 클래스를 포함하는 자바 소스 파일 구조

```
package・・・;           // 1) 패키지 : 프로젝트 생성 시 패키지를 지정했다면 이 구성 요소애 패키지명이 포함되며, 반드시 주석을 제외하고 첫 번째 줄에 위치
import ・・・;           // 2) 임포트 : 다른 패키지의 클래스를 사용하고자 할 때 포함된다.  패키지 다음에 위치한다
class 클래스명{・・・}   // 3) 외부 클래스 : 클래스의 외부에 또 다른 클래스가 또 포함 가능 , 1개의 자바 파일에 여러개의 클래스가 포함 가능, 외부 클래스에는 pulic
-------- 클래스 밖에 올수 있는 3가지 --------
-------- 클래스 안에 올수 있는 4가지 --------
public class betty {     //   public class 뒤에는 꼭 파일명과 동일해야 함 !
  int a = 3;             // 1) 필드 : 클래스의 특징(속성)을 나타내는 변수
  double abc( ) {・・・}  // 2) 메서드 : 클래스가 지니고 있는 기능(함수)
  betty() {・・・}        // 3) 생성자 : 클래스의 객체를 생성하는 역할
  class 클래스명{・・・}  // 4) 이너 클래스 : 클래스 내부에도 클래스가 포함 가능, 이를 "이너 클래스" 라고 한다
}
```

내부에 올 수 있는 4가지 구성 요소들 중 “필드” , “메서드” , “이너 클래스” 를 `클래스의 멤버` 라고 한다.

⇒ 클래스의 외부에는 3가지 종류 , 내부에는 4가지 종류만 올 수 있다. 이들 이외의 요소가 있으면 에러

❓ `접근 지정자` public 이란 ?

class 키워드 앞에 있는 public을 `접근 지정자` 라고 한다.

public 이 붙은 클래스 이름은 파일 이름과 동일해야 한다!