클래스(Class) 기초

클래스란

객체(Object)란?

- 사전적인 의미 => '실세계에 존재하거나 생각할 수 있는 것'.
- 실존 하는 책상, 볼펜 , 의자 등
- 강의, 수강신청과 같은 추상적인 개념도 객체라고 함.

객체지향프로그래밍(object oriented programming)이란?

- 객체의 사용을 통해 원하는 결과를 도출하는 프로그래밍 기법
- 프로그래머 관점에서의 객체 => 데이터와 기능를 변수(속성)와 메소드로 구현한 것

클래스(Class)란?

- 데이터(변수)과 기능(메소드)을 공유하는 유사한 성질의 객체들을 그룹화한 것.
- 클래스는 사용자 정의 자료형이다.
- 설계도와 비슷한 성격을 가지고 있다. 또는 공장이라고 생각해도 좋다.

클래스의 구성(1)

클래스는 변수(데이터), 메소드(기능), 생성자로 이루어져 있다.

- 클래스에서 사용하는 변수를 멤버변수 혹은 필드라고 부른다.
- 필드는 클래스 안에서 선언된 변수이다.
- 필드는 어떤 데이터의 값을 저장하고 보관하는 공간이 된다.
- 메소드는 메소드를 정의하는 부분과 호출하는 부분이 있다. 이 중 클래스를 구성하는 메소드는 메소드의 정의 부분을 말한다.
- 메소드는 멤버와 연관된 기능을 담당한다.
- 생성자는 클래스를 통해 인스턴스(객체)를 생성하게 해주는 역할을 한다.
- 생성자에서는 주로 멤버들을 초기화하는 작업을 한다.

클래스의 구성(2)

```
3  public class Student {
4      // 멤버
5      String name;
6      int age;
7
8      //생성자
9      public Student() {
10         name = "김자바";
11         age = 20;
12     }
13
14     //메소드의 정의(1)
15     public void setInfo(String name1, int age1) {
16         name = name1;
17         age = age1;
18     }
19     //메소드의 정의(2)
20     public void displayInfo() {
21         System.out.println("이름 : " + name);
22         System.out.println("나이 : " + age);
23     }
24 }
```

클래스의 사용 – 객체 생성

클래스, 인스턴스(객체), 참조변수

클래스 (붕어빵틀)

- 유사한 속성과 기능을 가진 객체들의 집합체. 그리고 그 객체를 만들기 위한 설계도

인스턴스 (붕어빵)

- 클래스에서 만들어진 실체
- 클래스를 통한 인스턴스 생성 방법 –> new 클래스명(); ex> new Student();
- new 키워드는 새로운 인스턴스(객체)를 생성하라는 의미이다

참조변수 (여러 개의 붕어빵 중 하나를 특정지을 수 있는 이름)

- 인스턴스를 컨트롤하기 위한 변수. 일반적인 변수라고 생각하면 된다.

클래스의 선언 및 인스턴스(객체) 생성

ex) Student student = new Student();
클래스 참조변수 인스턴스

클래스의 사용(필드 접근법)

```java
public class Student {
    // 멤버
    String name;
    int age;

    //생성자
    public Student() {
        name = "김자바";
        age = 20;
    }

    //메소드의 정의(1)
    public void setInfo(String name1, int age1) {
        name = name1;
        age = age1;
    }
    //메소드의 정의(2)
    public void displayInfo() {
        System.out.println("이름 : " + name);
        System.out.println("나이 : " + age);
    }
}
```

```java
public class StudentManager {
    public static void main(String[] args) {
        Student student = new Student();

        student.name = "이자바";
        student.age = 30;

        System.out.println("이름 : " + student.name);
        System.out.println("나이 : " + student.age);
    }
}
```

실행결과

```
이름 : 이자바
나이 : 30
```

- 객체이름.변수명으로 객체의 필드에 접근한다.
- 변수에 값을 할당하거나, 읽는 방법은 기본자료형 변수와 동일하다.

클래스의 사용(메소드 사용법1)

```java
public class Student {
    // 멤버
    String name;
    int age;

    //생성자
    public Student() {
        name = "김자바";
        age = 20;
    }

    //메소드의 정의(1)
    public void setInfo(String name1, int age1) {
        name = name1;
        age = age1;
    }
    //메소드의 정의(2)
    public void displayInfo() {
        System.out.println("이름 : " + name);
        System.out.println("나이 : " + age);
    }
}
```

```java
public class StudentManager {
    public static void main(String[] args) {
        Student a = new Student();
        a.setInfo("최자바", 40);
        a.displayInfo();
    }
}
```

실행결과

```
이름 : 최자바
나이 : 40
```

- 객체이름.메소드로 객체의 메소드를 사용

클래스의 사용(메소드 사용법2)

```
3 public class StudentManager {
4     public static void main(String[] args) {
5         Student a = new Student();
6         Student b = new Student();
7 
8         a.setInfo("최자바", 40);
9         a.displayInfo();
10 
11        b.setInfo("박자바", 50);
12        b.displayInfo();
13    }
14 }
```

실행결과

```
이름 : 최자바
나이 : 40
이름 : 박자바
나이 : 50
```

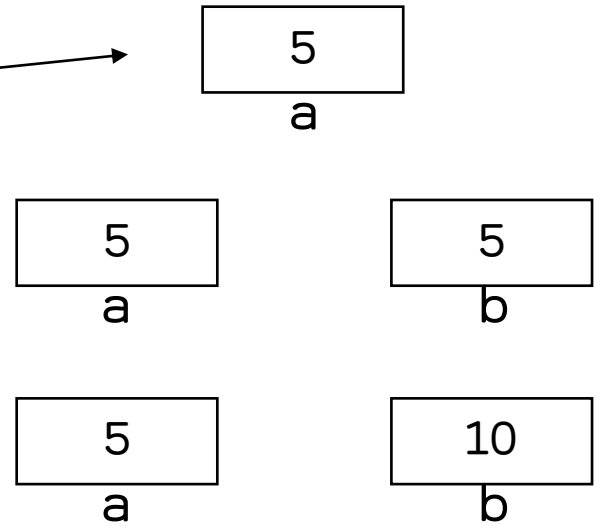
- 객체는 여러 개 만들 수 있다.
- 만들어진 객체는 다른 것이다. 즉 a와 b는 다른 객체이다.
- 그렇기 때문에 a 객체를 컨트롤 한다고 해서 b객체가 영향을 받지는 않는다.
- 이해를 위해 붕어빵을 생각하면 된다.
- 붕어빵 틀(클래스)에서 붕어빵(객체)을 여러 개 만들 수 있다.
- 붕어빵 하나를 슈크림으로 만들었다고 해서 모든 붕어빵의 속이 슈크림이 되는 것이 아니다.
- 붕어빵 하나를 한입 베어 먹었다고 해서, 다른 붕어빵도 갑자기 한 입 베어 진 것처럼 사라지는 것은 아니다.

변수 vs 참조변수(1)

변수

```
3 public class StudentManager {
4     public static void main(String[] args) {
5         int a = 5; //5의 값을 값는 a 생성
6         int b = a; //b를 생성
7
8         b = b + 5; //b값을 변경
9
10        System.out.println("a = " + a);
11        System.out.println("b = " + b);
12    }
13 }
```

```
a = 5
b = 10
```



변수에서는 대입연산자(=)를 통해 값을 대입하면 변수의 값이 복사된다

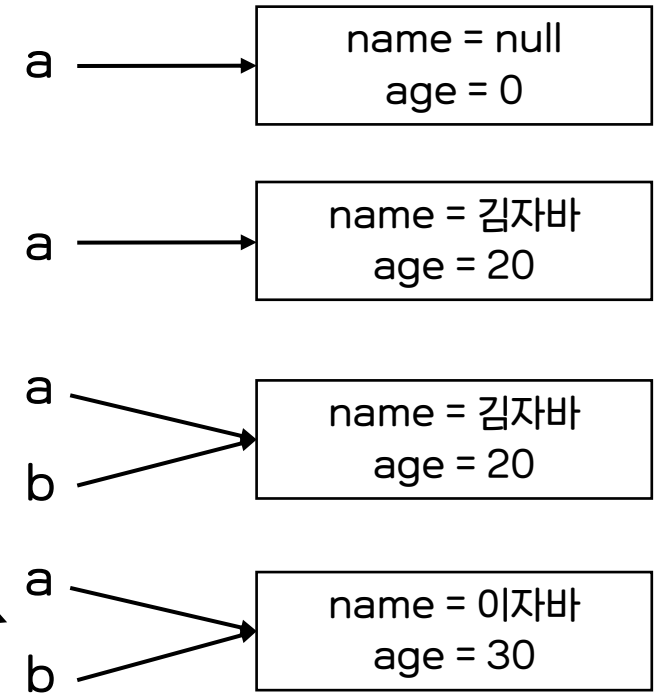
변수 vs 참조변수(2)

참조변수

```
3 public class StudentManager {
4     public static void main(String[] args) {
5         Student a = new Student(); //객체 a를 생성
6         a.setInfo("김자바", 20); //객체 a의 정보 변경
7
8         Student b = a; //객체 b를 생성
9         b.setInfo("이자바", 30); //객체 b의 정보 변경
10
11        a.displayInfo(); //객체 a의 정보 출력
12        System.out.println();
13        b.displayInfo(); //객체 b의 정보 출력
14    }
15 }
```

```
이름 : 이자바
나이 : 30

이름 : 이자바
나이 : 30
```



- 참조변수는 대입연산자를 통해 값을 대입하면 변수의 값을 가리키는 것이 두 개가 된다
- 이렇게 클래스의 객체를 변수에 저장하고 있는 경우, 그 값 자체를 저장하고 있지 않고 값이 저장된 공간을 참조하고 있기 때문에 참조변수라는 말을 쓴다.

생성자의 소개(1)

```
2  public class Account {
3      String owner; //계좌명의자
4      String accountNumber; //계좌번호
5      int balance; //예금액
6
7      public void initAccount(String owner1, String accountNumber1, int balance1) {
8          owner = owner1;
9          accountNumber = accountNumber1;
10         balance = balance1;
11     }
12
13     public void showAccountInfo() {
14         System.out.println("계좌명의자 : " + owner);
15         System.out.println("계좌번호 : " + accountNumber);
16         System.out.println("예금액 : " + 1000);
17     }
18 }
```

- 위는 계좌 클래스를 구현한 것이다.
- 계좌 클래스는 멤버로 계좌 명의자, 계좌번호, 예금액을 가지고 있다.
- 계좌 생성 후 계좌 정보를 초기화하는 initAccount() 메소드가 있다.
- 개설된 계좌의 정보를 확인 할 수 있는 showAccountInfo() 메소드도 존재한다.

생성자의 소개(2)

```java
public class Account {
    String owner; //계좌명의자
    String accountNumber; //계좌번호
    int balance; //예금액

    public void initAccount(String owner1, String accountNumber1, int balance1) {
        owner = owner1;
        accountNumber = accountNumber1;
        balance = balance1;
    }

    public void showAccountInfo() {
        System.out.println("계좌명의자 : " + owner);
        System.out.println("계좌번호 : " + accountNumber);
        System.out.println("예금액 : " + 1000);
    }
}
```

```java
public class AccountManager {
    public static void main(String[] args) {
        Account sinhan = new Account();

        //계좌정보를 초기화한다.
        sinhan.initAccount("김자바", "100-100", 1000);
        sinhan.showAccountInfo();
        System.out.println();
        //다시 계좌정보를 초기화한다?
        sinhan.initAccount("이자바", "100-200", 1000);
        sinhan.showAccountInfo();
    }
}
```

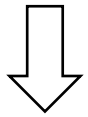
- AccountManager 클래스는 Account 클래스를 실행하기 위한 클래스다.
- 실제 코딩 후 실행 결과를 보면 문제가 없다.
- 하지만 논리적인 오류가 존재한다. 계좌 초기화 진행 후 다시 초기화가 되는 것이 맞는 걸까.
- 계좌를 하나 개설했는데 정보가 바뀐다는건 엄청 난감한 일이다.
- 그럼 initAccount() 메소드를 한번만 호출할 수 있도록 해야 하는데.. 그것이 가능할까..

클래스의 생성자(1)

잘 만들어진 클래스는 모두 멤버 초기화를 목적으로 하는 메소드가 있다. 하지만, 우리가 일반적으로 알고 있는 메소드를 사용하여 초기화를 한다면 언제든지 메소드를 호출할 수 있기 때문에 초기화라는 의미 자체가 무의미해진다. 이런 문제를 해결하기 위한 것이 바로 생성자의 사용이다. 결국, 생성자를 통해 우리는 멤버들의 값을 적절하게 초기화를 시켜줘야 한다. 그리고 생성자는 객체를 생성할 때 딱 1번만 호출할 수 있다. 그리고 다행스럽게 여러분들은 이미 생성자를 알게 모르게 사용해왔다.

먼저, Account 클래스의 initAccount() 메소드를 생성자로 바꾸어보겠다.

```
public void initAccount(String owner1, String accountNumber1, int balance1) {
    owner = owner1;
    accountNumber = accountNumber1;
    balance = balance1;
}
```



```
public Account(String owner1, String accountNumber1, int balance1) {
    owner = owner1;
    accountNumber = accountNumber1;
    balance = balance1;
}
```

- 메소드와 생성자는 매개변수로 원하는 값을 받아, 멤버의 값을 세팅하는 똑같은 기능을 한다.
- 그렇기 때문에 매개변수 및 내용은 동일하다.
- 생성자의 형태를 유심히 살펴보자

클래스의 생성자(2)

생성자의 형태

```
public 클래스명(매개변수들){
  //내용부
}
```

```java
public Account(String owner1, String accountNumber1, int balance1) {
    owner = owner1;
    accountNumber = accountNumber1;
    balance = balance1;
}
```

- 생성자가 모두 public으로 시작하는건 아니다.
- 하지만 지금은 모두 public으로 시작한다고 알고 있어도 된다.
- 생성자는 대소문자를 구분하여 정확히 클래스명과 일치해야 한다.
- 리턴형이 없다. 심지어 void도 적으면 안된다.
- 리턴이 없다. 리턴형이 없기 때문에 리턴이 없는건 당연한거다.
- 내용부에는 객체를 초기화하는 기능을 구현하면 된다.
- 생성자는 객체를 생성할 때 반드시 한 번 호출되어야 한다.

클래스의 생성자(3)

다음은 Account클래스에 생성자를 추가하고 실행한 예이다.

```java
public class Account {
    //멤버
    String owner;
    String accountNumber;
    int balance;

    //생성자
    public Account(String owner1, String accountNumber1, int balance1) {
        owner = owner1;
        accountNumber = accountNumber1;
        balance = balance1;
    }

    //멤버 정보 출력 메소드
    public void showAccountInfo() {
        System.out.println("계좌명의자 : " + owner);
        System.out.println("계좌번호 : " + accountNumber);
        System.out.println("예금액 : " + balance);
    }
}
```

```java
public class AccountManager {
    public static void main(String[] args) {
        Account sinhan = new Account("김자바", "100-100", 10000);
        sinhan.showAccountInfo();
    }
}
```

```
계좌명의자 : 김자바
계좌번호 : 100-100
예금액 : 10000
```

클래스의 생성자(4)

다시 한번 객체를 생성하는 방법을 살펴보자.

```
Account sinhan = new Account("김자바", "100-100", 10000);
```

① ② ③

①은 클래스에 대한 객체를 선언하는 부분이다. (int a와 같은 구조)

② new 키워드는 '새로운 인스턴스(객체)를 생성하라'는 의미이다.

③번 코드가 생성자를 사용한 부분이다. 이 코드에 의해 멤버의 정보가 초기화된 객체가 생성된다.

그럼, 처음에 보았던 Student 클래스의 객체 생성 부분을 보겠다.

```
Student student = new Student(); ①
```

- ①부분이 사실 생성자였던 것이다.
- 이렇게 객체를 생성할 때 생성자를 사용하기 때문에 초기화에 적절한 구조를 갖게 되었다.
- 하지만 여기의 의문점이 하나 생긴다. 우리는 Student 클래스에서 생성자를 만든 적이 없다.
- 심지어 Student클래스를 공부할 때 우리는 생성자가 무엇인지 알지도 못했다.

클래스의 생성자(5)

Student student = new Student();

설명했듯이 Student()는 생성자를 호출한 것이다. 우리가 생성자를 만들지 않았어도 생성자 호출과 객체 생성에는 문제가 없었다. 이는 생성자의 특징을 보면 알 수 있다.

생성자의 특징

- 클래스명과 동일하다.(대소문자도)
- 결과형 리턴값을 가지지 않는다.
- 객체가 생성될 때 반드시 하나의 생성자가 호출된다.
- 주로 객체 생성과 동시에 멤버 필드들의 값을 초기화하는 기능을 한다.
- **하나의 클래스에 생성자가 하나도 없다면 자동적으로 default 생성자가 있는 것으로 한다.**
  (default 생성자는 매개변수도 없고 내용도 없는 생성자를 말한다)
- 하나의 클래스에는 매개변수의 개수가 다르거나, 매개변수의 자료형이 다른 생성자가 여러 개 있을 수 있다.(생성자 오버로딩)

클래스의 생성자(6)

Student student = new Student();

즉, 위의 문장에서 객체 생성이 가능한 이유는 default 생성자를 호출했기 때문이다.

Student클래스에 default 생성자를 작성한다면 아래와 같다.

```
3  public class Student {
4      // 멤버
5      String name;
6      int age;
7
8      //default 생성자
9      public Student() {
10
11     }
12
13     //메소드의 정의(1)
14     public void setInfo(String name1) {
15         name = name1;
16         age = 50;
17     }
18
19
20     //메소드의 정의(2)
21     public void displayInfo() {
22         System.out.println("이름 : " + name);
23         System.out.println("나이 : " + age);
24     }
25 }
```

Default 생성자

- 생성자가 존재하지 않을 때 자동으로 만들어진다.
- 자동으로 만들어지지만 눈에는 보이지 않는다.
- 매개변수와 내용부가 비어있기 때문에 하는 기능이 없는 껍데기일 뿐이다.
- 즉, 단순히 객체 생성만 한다.

this 라는 키워드(1)

Student 클래스의 setInfo() 메소드를 살펴보자. 이 메소드는 매개변수로 이름과 나이를 받아 각각의 값을 세팅하는 기능을 한다.

```
3  public class Student {
4      // 멤버
5      String name;
6      int age;
7
8      //생성자
9      public Student() {
10         name = "김자바";
11         age = 20;
12     }
13
14     //메소드의 정의(1)
15     public void setInfo(String name1, int age1) {
16         name = name1;
17         age = age1;
18     }
19     //메소드의 정의(2)
20     public void displayInfo() {
21         System.out.println("이름 : " + name);
22         System.out.println("나이 : " + age);
23     }
24 }
```

메소드를 잘 만드는 것은 프로그래밍에서 아주 중요한 요소이다. 그리고 좋은 메소드인지 판단하기 위한 요소 중 하나가 매개변수명이다.

```
public void displayInfo(String aaa, String bbb, int ccc) {
    System.out.println(aaa + " , " + bbb + ", " + ccc);
}
```

위 메소드는 받은 매개변수를 출력하고 있다. 하지만 무엇을 출력하는지 가늠하기 어렵다.

```
public void displayInfo(String name, String address, int age) {
    System.out.println(name + " , " + address + ", " + age);
}
```

위 메소드도 받은 매개변수를 출력하고 있다. 하지만 매개변수 명을 바꾸었을 뿐인데, 우리는 무엇을 출력하는지 짐작할 수 있다.

this 라는 키워드(2)

다시 Student 클래스의 setInfo() 메소드를 보자.

```
//메소드의 정의(1)
public void setInfo(String name1, int age1) {
    name = name1;
    age = age1;
}
```

매개변수명으로 name1과 age1을 사용하였다. 매개변수명을 보면 어떤 값이 들어올지 대충 짐작이 간다. 하지만 변수명 뒤에 '1'이 붙어있어서 뭔가 찜찜하다.

이쯤에서 여러분들은 매개변수명으로 name과 age가 더 적절하다는 판단을 했을 것이다. 자, 그럼 setInfo() 메소드를 수정해보자. 그리고 Student의 객체를 만들어서 프로그램이 잘 돌아가는지 확인해보자.

```
//메소드의 정의(1)
public void setInfo(String name, int age) {
    name = name;
    age = age;
}
```

소스를 수정하면 위와 같이 노란줄이 생기면서 경고가 생긴다. 경고는 오류와 다르게 주의하라는 말이므로 프로그램을 실행하는 것에는 문제가 없을 것이다.

this 라는 키워드(3)

실행 결과 뭔가가 이상하다는 것을 확인 할 수 있다.

```
//메소드의 정의(1)
public void setInfo(String name, int age) {
    name = name;
    age = age;
}
```

이 메소드 때문에 문제가 발생하는 것인데 왜 문제가 발생하는 것일까.

name = name 부분을 살펴보자. 왼쪽에 있는 name은 Student클래스의 멤버인 name인가?

아니면 매개변수로 받은 name인가? Age도 마찬가지다.

우리는 변수의 스코프에 대해 배웠다. 필드 역시 변수이기 때문에 같은 제약을 받는다. 우리가 배운대로면 setInfo()안에 name이라는 매개변수명을 애초에 사용할 수 없다. 그런데 사용가능하다. 말이 길어지기 때문에 수업에서 설명하기로 하겠다.

결론적으로, 두 name은 모두 매개변수인 name을 가르킨다. 그렇기 때문에 이상한 것이다.

우리가 원하는 것은 멤버인 name에 매개변수인 name을 넣는 것이다. 코드로 표현하자면

name(멤버) = name(매개변수) 이다. 이 문제를 해결하기 위해 this라는 키워드가 존재한다.

this 라는 키워드(4)

Student 클래스를 this 키워드를 사용하여 다시 만들어 봤다.

```
3  public class Student {
4      // 멤버
5      String name;
6      int age;
7
8      //메소드 정의(1)
9      public void setInfo(String name, int age) {
10         this.name = name;
11         this.age = age;
12     }
13
14     //메소드 정의(2)
15     public void displayInfo() {
16         System.out.println("이름 : " + this.name);
17         System.out.println("나이 : " + this.age);
18     }
19 }
```

필드 앞에 this. 이라는 키워드를 붙여 사용한다. 주의할 점은 지역변수에는 this라는 키워드를 사용할 수 없다는 것이다.

this.name 을 직역하자면 '이것의 name' 정도이다. 여기서의 '이것'은 '이 클래스의' 로 다시 해석하는 것이 이해하기 쉬울 것이다. 다시 말해서 this.name 은 '이 클래스의 name'이라는 말이 되므로 멤버인 name을 가리킨다.

또한, this라는 키워드를 다른 곳에서도 사용한 것을 볼 수 있다. 솔직히 말하자면 this라는 키워드는 생략된 것이다. 그렇기 때문에 적어주지 않아도 자동으로 this.name으로 인식한다.

문제는 이전의 setInfo() 메소드처럼 매개변수로 같은 이름을 사용한 경우인데, 이 경우에는 명시적으로 무조건 this키워드를 사용해 주어야 한다.
Student 클래스를 왼쪽과 같이 수정 후 다시 프로그램을 실행시켜보면 오류가 없다는 것을 확인 할 수 있을 것이다.

이제 우리는 this라는 키워드를 사용함으로써, 보다 멋진? 메소드를 작성할 수 있게 되었다.