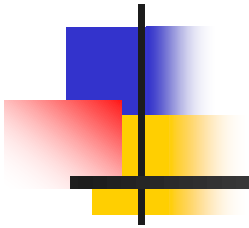
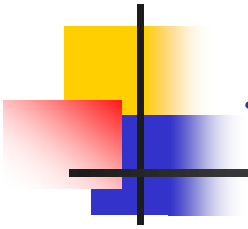


Java 5강-클래스

양 명 숙
[now4ever7@gmail.com]



목차

- 객체 지향 프로그래밍
- 클래스
  - 객체 생성
    - 메모리 할당/생성자 호출
- 생성자
  - 기본 생성자/ 재정의 생성자
- 접근 제한자
  - public, private, default, protected
- getter/setter
- this

객체지향 프로그래밍

- 객체(Object) 객체는 상태정보와 행동으로 이루어져있음
  - 물건, 대상
  - 주변에 존재하는 물건(컴퓨터, 책상, 휴대폰, 사과 등)이나 대상(친구, 선생님, 철수 등) 전부를 의미함
- 객체 지향 프로그래밍
  - 현실에 존재하는 사물과 대상, 그리고 그에 따른 행동을 있는 그대로 실체화시키는 형태의 프로그래밍
  - 예) 나는 과일장사에게 두 개의 사과를 구매했다.
    - 객체의 종류 – 나, 과일장사, 사과
    - '나'라는 객체가 '과일장사'라는 객체로부터 '사과'객체를 구매하는 행위도 그대로 표현할 수 있다.
  - 예) 나는 은행계좌에서 30000원을 출금했다.

객체를 이루는

예) **사람** – 상태정보 : 이름, 피부색, 키, 눈, 코, 입
행동 : 말하다, 걷다, 숨쉬다, 보다

TV - 데이터: 전원상태, 크기, 길이, 높이, 색상, 볼륨, 채널
기능: 켜기, 끄기, 볼륨 높이기, 채널 변경하기

객체를 이루는 것은 데이터와 기능

- 객체
  - 상태 정보(속성, 데이터)와 행동(기능)으로 구성됨
  - 상태정보 => 변수를 통해서 표현됨
  - 행동 => 메소드를 통해 표현됨
- 예) '나'와 '과일장사'라는 객체를 생성하여 다음의 행동을 실체화시키자
  - => 나는 과일장사에게 2000원을 주고 두 개의 사과를 구매했다.
  - 과일장사 객체
    - 과일장사는 과일을 판다. (행동)
    - 과일장사는 사과 20개, 귤 10개를 보유하고 있다. – 과일의 개수 (상태)
    - 과일장사의 과일 판매수익은 50000원이다. – 판매수익 (상태)
- 예) => 나는 은행계좌에서 30000원을 출금했다.
  - 은행계좌 객체
    - 은행에서 출금한다/ 입금한다 (행동)
    - 계좌번호, 이름, 잔액 (상태)

예1) 은행계좌

• 은행 계좌 객체
은행에서 출금한다/ 입금한다 (행동)
계좌번호, 이름, 잔액 (상태)

- 은행계좌의 상태정보를 변수로 표현
  - 계좌번호 => String accId
  - 이름 => String name
  - 잔액 => int balance
- 출금하는 것을 메소드로 표현

```
public void withdraw(int money)  //출금
{
    balance -= money;
}
```

- 은행계좌 객체를 구성하는 변수와 메서드를 묶어서 객체라는 것을 통해서 실체화하면 됨

예2) 과일 장사

• 과일장사 객체
과일장사는 과일을 판다. (행동)
과일장사는 사과 20개, 귤 10개를 보유하고 있다.
- 과일의 개수 (상태)
과일장사의 과일 판매수익은 50000원이다.
- 판매수익 (상태)

- 과일장사의 상태정보를 변수로 표현
  - 보유하고 있는 사과의 수 => int numOfApple
  - 판매수익 => int myMoney
- 과일의 판매를 메소드로 표현

```
int saleApple(int money){ //과일 구매액
    int num=money/1000; //사과가 개당 1000원
    numOfApple -= num;   //사과의 수가 줄어들고,
    myMoney += money;  //판매수익이 발생
    return num; //실제 구매가 발생한 사과의 수
}
```

- 과일장사 객체를 구성하는 변수와 메서드를 묶어서 객체라는 것을 통해서 실체화하면 됨

클래스라는 틀을 기반으로 객체가 생성

- 객체를 생성하기 전에 객체의 생성을 위한 '틀'을 먼저 만들어야 함
  - 붕어빵을 만들기 위해서는 붕어빵틀이 필요
  - 자동차를 만들기 위해 자동차의 엔진과 외형을 생산할수 있는 틀(제품설계도)이 필요
  - '은행계좌' 객체를 생성하기 위해서는 이를 위한 틀을 먼저 만들어야 함

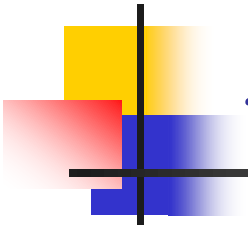
  - 클래스의 정의

```
class Account
{
    //멤버 변수 선언
    String accId;   //계좌번호
    String name;    //이름
    int balance;    //잔액

    //멤버 메서드 선언 (출금하다)
    public void withdraw(int money)
    {
            balance -= money;
    }
}
```

Account 라는 이름의 틀을 정의
⇒ 이러한 틀을 가리켜 클래스(Class)라 함
⇒ 클래스는 실체(객체)가 아닌 틀
⇒ 클래스는 객체를 구성하는데 필요한 변수와 메소드로 이뤄짐
⇒ 클래스 안에 정의된 메서드 내에서는 동일한 클래스 안에 선언된 변수에 접근이 가능함

클래스	객체
제품설계도	제품
자동차 설계도	자동차
붕어빵틀	붕어빵

틀 실체



클래스 정의

- 코드화시킨 클래스
  - 필드 – 클래스의 멤버 변수
  - 메서드 – 클래스의 멤버 함수

- 클래스 정의하기

```
class 클래스 이름
{
    멤버변수;
    메서드();
}
```

Account 클래스

클래스를 작성한 다음, 클래스로부터 객체를 생성하여 사용

객체를 사용한다는 것 - 객체가 가지고 있는 속성과 기능을 사용한다는 뜻

```
class Account
{
	//멤버 변수 선언
	String accId;   //계좌번호
	String name;    //이름
	int balance;    //잔액

	//멤버 메서드 선언
	public void deposit(int money)   //입금하다
	{
		balance += money;
	}
	public void withdraw(int money)  //출금하다
	{
		balance -= money;
	}
	public void showBalance()  //추가된 메서드
	{
		System.out.println("==========계좌 정보 ==========");
		System.out.println("계좌번호 : "+ accId);
		System.out.println("이름 : "+ name);
		System.out.println("잔액 : "+ balance);
	}
}//Account
```

main()에서 Account객체 생성

```
public class AccountMain{
    public static void main(String[] args)   {
        //1. 객체 생성 - 해당 클래스의 멤버변수와 메서드를 메모리에 할당
        Account acc;
        acc = new Account();

        //2. 메서드 사용
        acc.showBalance();  //멤버변수는 자동으로 디폴트값으로 초기화됨

        //두번째 객체(또 다른 객체) 생성
        Account acc2=new Account();

        //멤버변수 사용
        acc2.accId="100-2000-001";
        acc2.name="홍길동";
        acc2.balance=100000;

        //메서드 사용
        acc2.withdraw(30000);  //30000원 출금
        acc2.showBalance();

        acc2.deposit(50000);  //50000원 입금
        System.out.println("현재 잔액 : "+acc2.balance);  //멤버변수 사용
    }
}
```

```
=======계좌정보=======
계좌번호 : null
이름 : null
잔액 : 0
=======계좌정보=======
계좌번호 : 100-2000-001
이름 : 홍길동
잔액 : 70000
현재 잔액 : 120000
```

클래스를 기반으로 객체 생성하기

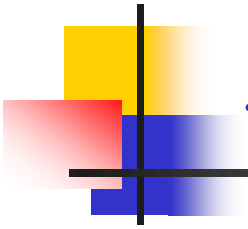
- 클래스는 실체(객체)가 아닌 '틀'이므로 클래스 안에 존재하는 변수에 접근하고, 메소드를 호출하기 위해서는 클래스를 실체화(객체화)시켜야 함
- 객체 생성 방법

```
int  a;  //변수 선언
int[]  arr = new int[3]; //배열선언, 메모리 할당
```

클래스이름 변수명 = **new** 클래스이름( );

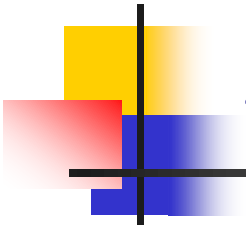
- Account acc = new Account();
  - Account 객체를 생성하고 이를 acc라는 이름의 변수로 참조하는 문장
  - acc 를 통해서 각각의 객체에 접근할 수 있게 됨
  - acc => 참조변수

```
Scanner sc = new Scanner(System.in);
```



생성된 객체의 접근방법

- 객체의 변수(멤버변수)에 값 저장
  - Account acc = new Account();
  - acc.balance=100000;
- 객체의 메서드 호출
  - acc.withdraw(30000);
- . 연산자를 이용해서 객체의 변수나 메서드에 접근



클래스를 기반으로 객체 생성하기

- new
  - 객체 생성을 명령하는 명령어
  - 메모리 공간에 객체가 생성됨
  - 클래스로부터 객체를 만드는 과정을 클래스의 '인스턴스화(instantiation)'라고 함
  - 어떤 클래스로부터 만들어진 객체를 그 클래스의 '인스턴스(instance)'라고 함

- 객체와 인스턴스
  - 객체 – 모든 인스턴스를 대표하는 포괄적인 의미
  - 인스턴스 – 어떤 클래스로부터 만들어진 것인지를 강조하는 보다 구체적인 의미
    - 예) 책상은 객체다.
    - 책상은 책상 클래스의 인스턴스다.

클래스를 기반으로 객체 생성하기

클래스

틀

```
class Account
{
   String accId;
   String name;
   int balance;

   public void deposit(int money)
   { ...
   }
   public void withdraw(int money)
   { ...
   }
   public void showBalance()
   { ...
   }
}
```

인스턴스화

new

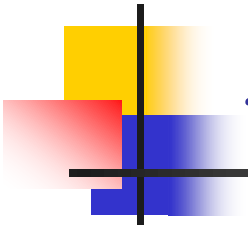
객체(인스턴스)

실체

```
String accId;
String name;
int balance;

public void deposit(int money)
{ ...
}
public void withdraw(int money)
{ ...
}
public void showBalance()
{ ...
}
```

변수

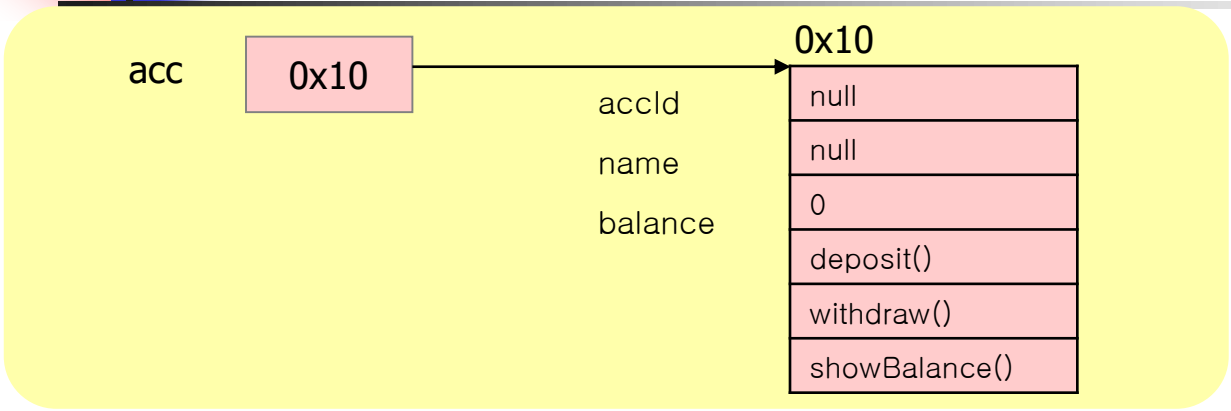
메서드



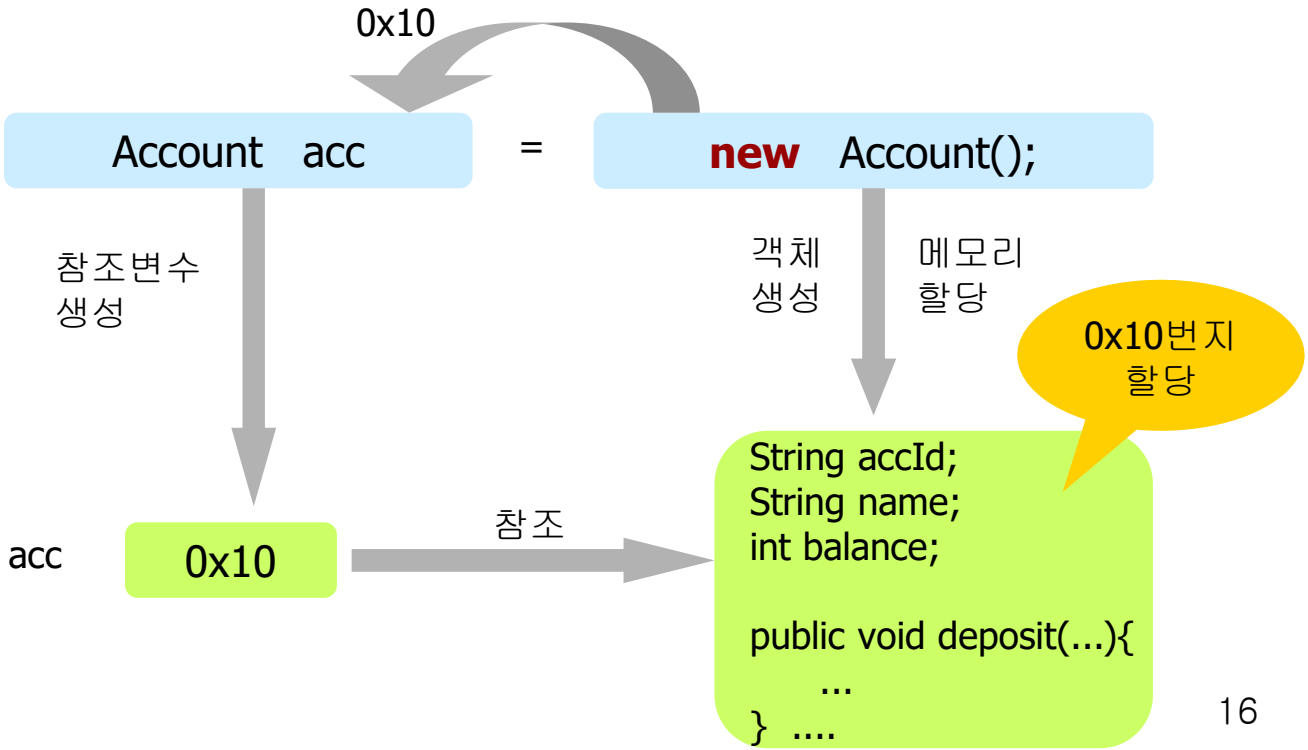
클래스를 기반으로 객체 생성하기

- 클래스
  - 클래스에 존재하는 변수와 메소드는 메모리 공간에 할당된 형태로 존재하지 않음
  - 접근도 호출도 불가능한, 하나의 틀로서만 역할을 함
- 객체
  - **메모리 공간에 할당이 이뤄짐**
  - 객체를 구성하는 모든 변수는 그 크기대로 메모리 공간에 할당이 되고, 메소드도 호출할 수 있는 형태로 메모리 공간에 존재하게 됨

객체 생성과 참조의 관계




```
acc.name = "홍길동";
System.out.print(acc.name);
```



```
acc.name="홍길동";
acc.withdraw();
```

값타입과 참조타입의 메모리 생성 영역

Stack 메모리 영역

a **5**

acc 0x101

```
int a;
a=5;
```

```
Account   acc;
acc=new Account();
```

Heap 메모리 영역

0x101

accId null

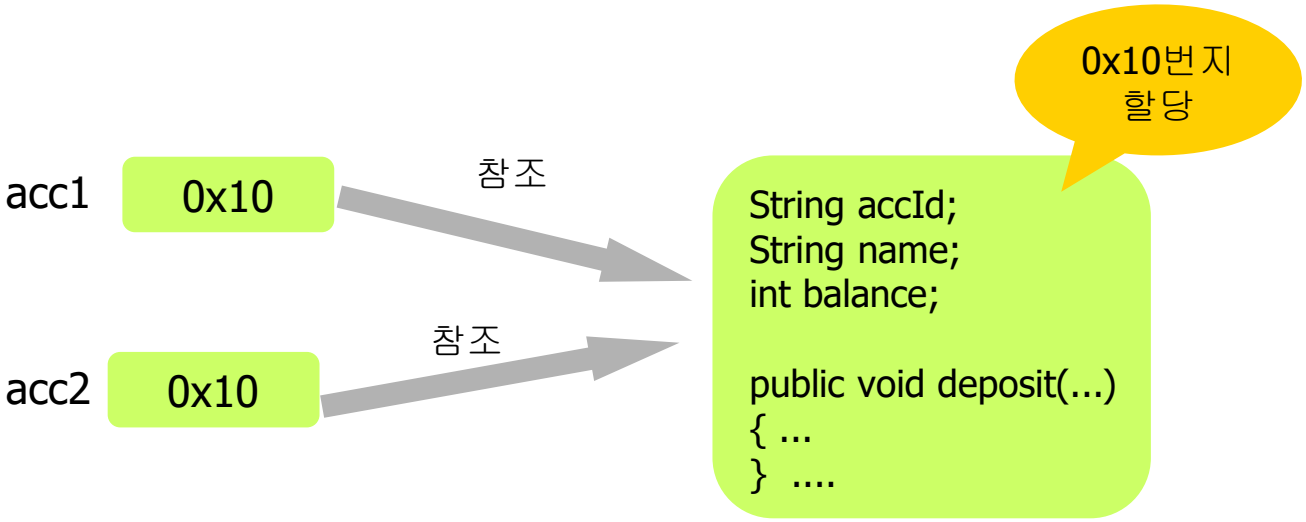
name null

withdraw()

```
class Account{
    String accId;
    String name;

    public void withdraw(){…}
}
```

객체 생성과 참조의 관계

- 키워드 new에 의한 객체 생성시 생성된 객체는 메모리에 저장되고, 저장된 메모리의 주소 값이 반환되어 참조변수에 저장됨
  - **참조변수에는 주소 값이 저장**되므로, 참조변수에 의한 객체 접근이 가능
  - Account acc1 = new Account();
  - Account acc2 = acc1;
    - 두 개의 참조변수가 하나의 객체를 참조하게 됨



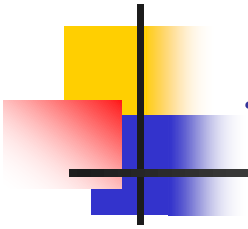
클래스를 기반으로 객체 생성하기

- 클래스를 정의하는 것은 자료형을 정의하는 것임
  - 새로운 타입의 자료임
  - 기본 자료형이 아닌 사용자 정의 자료형(참조 타입)
  - 서로 관련된 변수들을 정의하고, 이들에 대한 작업을 수행하는 메서드들을 함께 정의하는 것

```
int  a;  //기본자료형, 값타입

int[]  arr = new int[3]; //참조타입
```

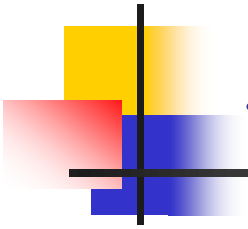
클래스
- 데이터와 메서드의 결합

```
Account   acc = new Account();   //참조타입

Account   acc;
acc = new Account();
```



복습 – 데이터타입(Data Type)

- 데이터타입 분류
  - **값타입(Value Type)**
    - 기본 자료형 - byte, short, int, long, float, double, char, boolean
  - **참조타입(Reference Type)**
    - 기본 내장형 – Object 형, String형
    - 클래스, 인터페이스, 배열 등



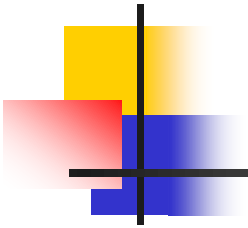
객체 지향 언어

- 객체 지향 언어
  - 소스코드의 재사용이 쉽고, 실세계와도 유사한 개념인 객체로 프로그램을 구성한 것
  - 필요한 기능을 **부품**처럼 미리 만들어 놓고, 필요한 경우 바로 사용하여 구현
  - 기존의 구조적 프로그래밍에서는 변수와 함수를 어느 특정 객체에 포함시키지 않고 사용했고, 객체 지향언어에서는 특정객체에 맞는 변수와 함수를 만들어 제공해야 함

예제

```
원의 반지름을 입력하세요
7
원의 넓이 : 153.86
원의 둘레 : 43.96
```

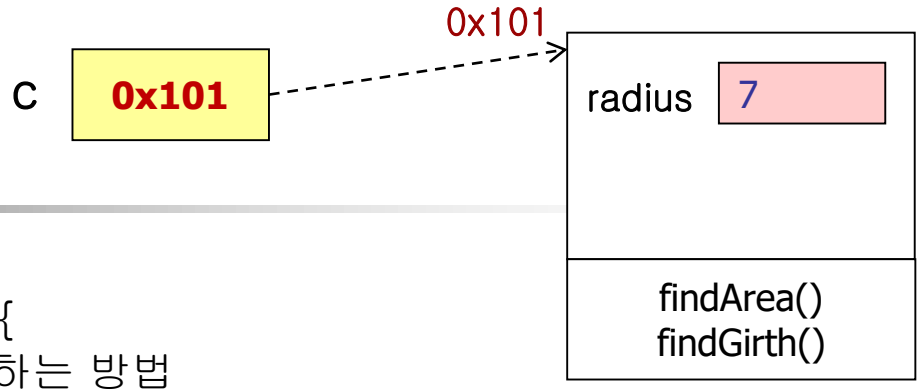
- 원을 나타내는 Circle 클래스 디자인
  - 반지름 멤버 변수
  - 넓이를 구하는 기능
  - 둘레를 구하는 기능
- 메인 클래스의 main() 메서드에서
  - 사용자에게 반지름을 입력 받고
  - Circle Class의 메서드를 호출하여 넓이와 둘레를 구한 후 화면에 출력하기



예제

```
class Circle
{    //원을 나타내는 클래스
     //1. 멤버변수 - 상태정보, 특징
     int radius; //반지름

     //2. 메서드-행동, 기능
     //원의 넓이를 구하는 메서드
     public double findArea()
     {
          //넓이 = 반지름*반지름*3.14
          double area =  radius*radius*3.14;
          return area;
     }
     //원의 둘레를 구하는 메서드
     public double  findGirth()
     {
          //둘레 = 2*반지름*3.14;
          double girth = 2*radius*3.14;
          return girth;
     }
}//class
```

예제-계속




```
class CircleTest {
    public static void main(String[] args) {
        //[1] 멤버변수에 값을 직접 할당하는 방법
        //객체 생성
        Circle c = new Circle();

        //멤버변수인 반지름 값 할당
        c.radius=7;

        //메서드 호출 - 원의 면적 구하기
        double area = c.findArea();
        System.out.println("반지름 : " + c.radius);
        System.out.println("원의 넓이 : " + area);

        //원의 둘레 구하기
        double girth = c.findGirth();
        System.out.println("원의 둘레 :" + girth);
    }
}//class
```

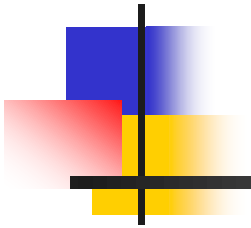
실습1 – 클래스 디자인

```
두실수를 입력하세요
10
20
a=10.0,b=20.0
a+b = 30.0
a-b = -10.0
a*b = 200.0
a/b = 0.5
```

- 1. 계산기 기능을 하는 Calculator 클래스 작성하기
  - 기본 기능 – 더하기, 빼기, 곱하기, 나누기
    - 두 실수를 매개변수로 갖는 메서드

- 2. 직사각형을 나타내는 Rectangle클래스 디자인
  - 멤버변수(필드) – 가로, 세로
  - 넓이를 구하는 기능
  - 둘레를 구하는 기능

```
사각형의 가로, 세로를 입력하세요
10
5
사각형의 넓이 : 50
사각형의 둘레 : 30
```



정리

추상화

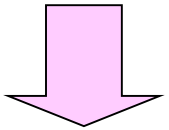
-주어진 문제나 시스템 중에서 **중요하고 관계 있는 부분만을 분리**하여 간결하고 이해하기 쉽게 만드는 작업.
-필요한 부분만을 표현할 수 있고 불필요한 부분을 제거하여 간결하고 이해하기 쉽게 만드는 작업.

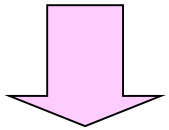
- 추상화(Abstraction)
  - 현실세계의 사물을 데이터적인 측면과 기능적 측면을 통해서 정의하는 것
  - 현실 세계의 어떠한 현상을 클래스화시키는 작업
  - 사물을 바라보고 행위(기능)와 특성을 구분해내는 작업
  - 무엇이 중요한지, 중요하지 않은 것이 무엇인지 결정
  - 가장 중요한 것에 포커스, 중요하지 않은 것을 무시
  - 중요한 것을 추출해서 코드로 옮겨 놓으면 class가 됨
  - 추상화 과정을 거쳐 클래스로 표현

예) 계좌
데이터(특성) : 계좌번호, 비밀번호, 이름, 잔액
기능(행위) : 입금하다, 출금하다

추상화

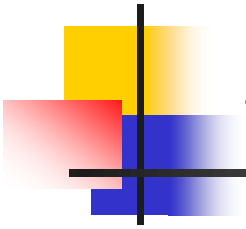
사물	상태(속성)	행동(기능)
사람	피부색, 키, 나이, 성별, 이름	생각한다, 공부한다, 말한다, 걷는다
차	배기량, 차종, 연료의 종류	달린다, 멈춘다, 고장난다, 짐을 싣는다
노트북	CPU, 액정크기, 하드디스크 용량	부팅한다, 충전한다, CD-ROM을 읽는다
윈도우	크기, 위치, 배경색, 아이콘	이동한다, 최대화한다, 숨는다
게시판	글쓴이, 제목, 내용, 날짜	글쓰기, 수정하기, 삭제하기,목록보기, 상세보기





변수 (필드)

메서드



클래스(Class)

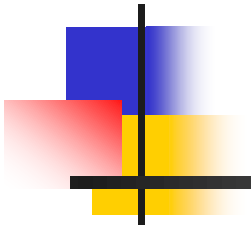
- 클래스(class)
  - 객체를 만들어 내는 설계도
  - 행위, **기능**(메서드) + **특성**, 데이타(변수)
  - Object를 행위와 특성으로 바라보고 그것을 코드로 옮겨 놓은 것
  - 행위와 특성을 syntax로 구조화시키는 것

- 서로 관련된 **변수**들을 정의하고, 이들에 대한 작업을 수행하는 **메서드**들을 함께 정의한 것
- 클래스를 정의한다는 것은 **자료형을 하나 정의**하는 것

객체(object)

- 객체(object)
  - 클래스를 정의한다는 것은 자료형을 하나 정의하는 것
  - 클래스를 이용해서 자료형을 정의하였으면 프로그램내에서 자료형을 기반으로 변수를 생성해야 함
  - 클래스가 메모리에 load된 것
    - 클래스는 코드, 이 코드를 메모리에 띄운 것이 객체(object)
  - 클래스는 직접 사용할 수 없고 클래스를 통해 객체를 생성해야만 그 객체를 사용할 수 있음

```
예)
    public static void main(String[] args)
    {
        Account   acc = new Account();
    }
```



생성자

생성자(Constructor)

- 생성자
  - 객체가 생성될 때 자동 호출되어 가장 먼저 실행되는 메서드
  - **멤버변수의 초기화를 목적**으로 정의되는 메서드
  - new 연산자가 힙(Heap)영역에 메모리를 생성한 직후 호출됨
  - 객체의 생성과 동시에 해주어야 하는 작업을 생성자에서 할수 있음
- 생성자가 되기 위한 조건
  - **클래스의 이름과 동일**한 이름의 메서드
  - 매개변수는 가질 수 있으나 **반환값은 가질 수 없음**
- 객체 생성 문장에는 호출될 생성자를 명시하는 부분이 존재함

```
클래스이름  변수이름  =  new  클래스이름( [매개변수] );
```

생성자(constructor) 메서드

  - Account acc = new Account();
  - new의 오른편에 있는 부분

객체 생성 순서

- Step 1: 메모리 할당
  - new 연산자를 사용하여 heap으로 부터 메모리 할당
- Step 2: 생성자를 이용한 객체 초기화
  - 클래스 이름을 메서드처럼 () 를 이용하여 명시

```
Account   acc =  new  Account();
```

기본 생성자(Default Constructor)

- 생성자 – 객체가 생성될 때 가장 먼저 호출되는 메서드
  - 1. 기본 생성자(default 생성자)
    - 따로 정해주지 않았을 때 기본으로 제공되는 생성자
      - 컴파일러에 의해 자동으로 만들어짐
    - 매개변수를 가지지 않는 생성자
    - 다른 생성자가 있으면 컴파일러에 의해 default 생성자가 만들어지지 않음
  - 2. 재정의 생성자(매개변수가 있는 생성자)
    - 사용자가 임의로 다시 만들어 놓은 생성자

컴파일러가 자동적으로 기본 생성자를 추가해주는 경우는 '클래스 내에 생성자가 하나도 없을 때' 뿐이다

기본 생성자 사용

- 기본 생성자의 특징
  - 클래스명과 동일한 이름 사용
  - **반환값도 없고, void도 아님**
  - 매개변수가 없음
  - 모든 필드를 0 (숫자필드), false(논리형), null(참조형)로 초기화
- 생성자 문법

```
class Account
{
    Account( )
    {
        ...
    }
}
```

예제

```
=========계좌 정보 =========
계좌번호 : null
이름 : null
잔액 : 0
```

```
public class AccountMain2
{
	public static void main(String[] args)
	{
		Account acc = new Account();//메모리할당과 객체 초기화를 동시에 처리, 기본생성자 사용

		acc.showBalance(); //기본 생성자에 의해 null, 0 으로 초기화
	}
}//

class Account
{
	String accId;   //계좌번호
	String name;    //이름
	int balance;    //잔액

	public void showBalance()
	{
		System.out.println("=========계좌 정보 =========");
		System.out.println("계좌번호 : "+ accId);
		System.out.println("이름 : "+ name);
		System.out.println("잔액 : "+ balance);
	}
}//Account
```

```
public Account()  //기본 생성자
{
		this.accId = null;
		this.name = null;
		this.balance = 0;
}
```

예제-생성자

```
public class AccountMain3{
	public static void main(String[] args)
	{
		Account acc = new Account();//재정의된 생성자 사용

		acc.showBalance();
	}
}//

class Account{
	String accId;   //계좌번호
	String name;    //이름
	int balance;    //잔액

	//생성자
	public Account()  //생성자 재정의
	{
		this.accId = "100-01-1000";
		this.name = "기본계좌";
		this.balance = 1000;
	}
	....
```

```
=========계좌 정보 =========
계좌번호 : 100-01-1000
이름 : 기본계좌
잔액 : 1000
```

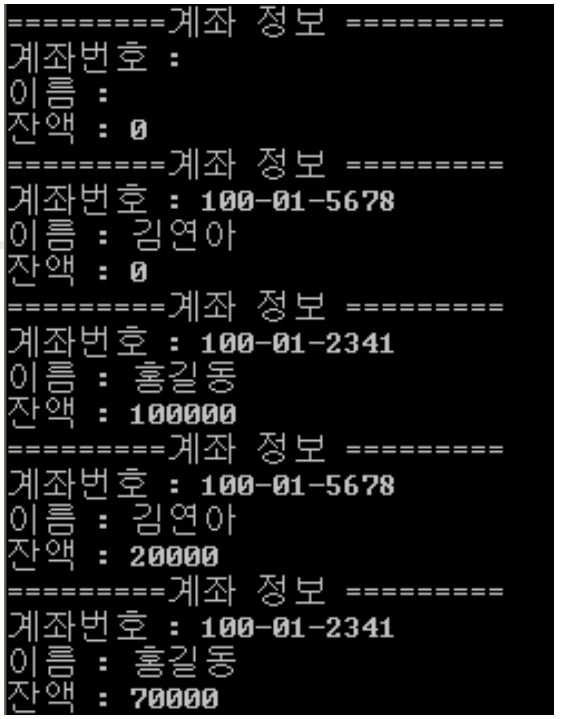
예제-생성자 오버로딩

```
========계좌 정보 ========
계좌번호 :
이름 :
잔액 : 0
========계좌 정보 ========
계좌번호 : 100-01-5678
이름 : 김연아
잔액 : 0
========계좌 정보 ========
계좌번호 : 100-01-2341
이름 : 홍길동
잔액 : 100000
```

```
class Account
{
    //멤버 변수 선언
    String accId;   //계좌번호
    String name;    //이름
    int balance;    //잔액

    public Account() //(1)생성자
    {
    }                   //이게 있어야만 빈 생성자 Account acc = new Account(); 해도 에러가 안 난다(매개변수 없는 생성자

    public Account(String accId, String name, int balance)//(2)생성자
    {
        this.accId = accId;
        this.name = name;
        this.balance = balance;
    }

    public Account(String accId, String name)//(3)생성자
    {
        this.accId = accId;
        this.name = name;
    }
```

예제-생성자 오버로딩

```
public class AccountMain4{
    public static void main(String[] args){
        //default 생성자가 만들어지지 않으므로,
        //매개변수 없는 생성자를 직접 만들어야 함
        Account acc1 = new Account();
        acc1.showBalance();

        Account acc2 = new Account("100-01-5678", "김연아");
        acc2.showBalance();

        Account acc3 = new Account("100-01-2341", "홍길동", 100000);
        acc3.showBalance();

        acc2.deposit(20000);
        acc2.showBalance();

        acc3.withdraw(30000);
        acc3.showBalance();

        acc2.balance=60000;
        acc2.name="김길동";
        acc2.showBalance();
    }
}//
```




```
Account   acc = new Account();
acc.accId = "100-01-2341";
acc.name = "홍길동";
acc.balance = 100000;
```

예제2 – 생성자 오버로딩

```
당신의 나이는 스물 셋 입니다.
당신의 나이는 23 입니다.
```

```
public class ConstructTest4
{
	public static void main(String[] args)
	{
		PrintAge Kor = new PrintAge("스물 셋"); //객체를 선언하고 힙에 할당
		PrintAge Num = new PrintAge(23);
		//PrintAge obj = new PrintAge();   //에러 - 디폴트생성자는 만들어지지 않으므로 사용불가
	}
}

class PrintAge
{
	public PrintAge(String age)   //(1)생성자
	{
		System.out.println("당신의 나이는 " + age + "입니다.");
	}

	public PrintAge(int age)     //(2) 생성자 - 매개변수의 타입이 다르다
	{
		System.out.println("당신의 나이는 "+age+" 입니다.", );
	}
}
```

비교 – 메서드 오버로딩

```
public class OverloadCalc {
	public int Plus(int a, int b) {
		return(a+b);
	}
	public float Plus(float a, float b) {
		return(a+b);
	}
	public double Plus(double a, double b) {
		return(a+b);
	}
	public static void Main() {
		OverloadCalc oc=new OverloadCalc();
		int i=oc.Plus(3,5);
		float j=oc.Plus(0.1f,0.2f);
		double k=oc.Plus(0.5,0.7);
		System.out.println("int합:“ + i);
		System.out.println("float합:“ + j);
		System.out.println("double합:“ + k);
	}
}
```

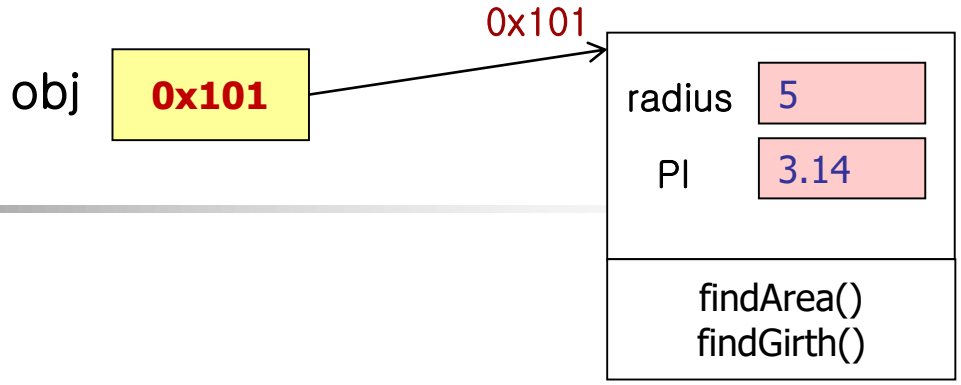
예제2

- 원을 나타내는 Circle 클래스 디자인
  - 반지름 멤버 변수
  - 넓이를 구하는 기능
  - 둘레를 구하는 기능
  - => 생성자를 이용해서 반지름 초기화
- 메인 클래스의 main() 메서드에서
  - 사용자에게 반지름을 입력 받고
  - Circle Class의 메서드를 호출하여 넓이와 둘레를 구한 후 화면에 출력하기

예제2

```
public class Circle{
	double radius;
	final double PI = 3.14;  //final 상수 선언

	//2. 생성자
	Circle()  //기본 생성자
	{
	}

	Circle(double r)  //매개변수가 있는 생성자
	{
		radius=r;			//멤버변수인 radius값을 초기화함
	}

	public double findArea()	// 넓이 구하기
	{
		return radius * radius * PI;
	}

	public double findGirth()	// 둘레 구하기
	{
		return  2 * radius * PI;
	}
}
```

예제2-계속




```
import java.util.*;
public class CircleTest
{
    public static void main(String[] args)
    {
        System.out.println("원의 반지름을 입력하세요");
        Scanner sc = new Scanner(System.in);
        double r = sc.nextDouble();  //5

        //[2] 생성자를 이용해서 멤버변수의 값을 할당(초기화)하는 방법
        //Circle obj = new Circle(r);
        Circle obj;
        obj = new Circle(r);

        System.out.println("원의 넓이 : " + obj.findArea());
        System.out.println("원의 둘레 : " + obj.findGirth());

        //멤버변수의 값을 변경할 수도 있다
        obj.radius = 30;
        System.out.println("원의 면적:" + obj.findArea());
    }
}
```

클래스의 객체를 참조하기 위한 참조변수를 선언

클래스의 객체를 생성 후, 객체의 주소를 참조변수에 저장

예제-삼각형 면적 구하기

```
import java.util.*;
class  TriangleTest1
{
    public static void main(String[] args)
    {
        //삼각형의 면적 구하기
        //[1] main() 에서 직접 면적을 구한다

        //사용자로 부터 입력 받기
        Scanner sc = new Scanner(System.in);
        System.out.println("삼각형의 밑변, 높이를 입력하세요");
        int w = sc.nextInt();
        int h = sc.nextInt();

        //삼각형의 면적구하기
        int area = w*h/2;

        //출력하기
        System.out.println("밑변 : "+ w +", 높이 :" + h +", 면적:"+area);
    }
}
```

```
import java.util.*;
class  TriangleTest2{
    //삼각형의 면적 구하는 메서드
    public static int findArea(int w, int h){
        int area = w*h/2;
        return area;
    }

    public static void main(String[] args) {
        //삼각형의 면적 구하기
        //[2] 메서드만 만들어서 메서드를 호출하여 면적을 구한다

        //사용자로 부터 입력 받기
        Scanner sc = new Scanner(System.in);
        System.out.println("삼각형의 밑변, 높이를 입력하세요");
        int w = sc.nextInt();
        int h = sc.nextInt();

        //삼각형의 면적구하기
        int area= findArea(w, h);

        //출력하기
        System.out.println("밑변 : "+ w +", 높이 :" + h +", 면적:"+area);
    }
}
```

```
import java.util.*;
class Triangle{
	//멤버변수가 없는 클래스
	//삼각형의 면적 구하는 메서드
	public int findArea(int w, int h)
	{
		int area = w*h/2;
		return area;
	}
}
class Triangle2{
	//멤버변수
	int width; //밑변
	int height; //높이

	//생성자
	Triangle2(int w, int h){
		width=w;
		height=h;
	}
	//삼각형의 면적 구하는 메서드
	public int findArea(){
		int area = width*height/2;
		return area;
	}
}
```

```
class  TriangleTest3{
    public static void main(String[] args)   {
        //삼각형의 면적 구하기
        //[3] class를 새로 만들어서 면적 구하는 메서드를 만들고, 이를 이용하여 면적을 구한다

        //사용자로 부터 입력 받기
        Scanner sc = new Scanner(System.in);
        System.out.println("삼각형의 밑변, 높이를 입력하세요");
        int w = sc.nextInt();
        int h = sc.nextInt();

        //삼각형의 면적 구하기
        //객체 생성
        Triangle t = new Triangle();
        //메서드 호출
        int area= t.findArea(w, h);

        //출력하기
        System.out.println("밑변 : "+ w +", 높이 :" + h +", 면적:"+area);

        //객체 생성
        Triangle2 tr = new Triangle2(w, h); //생성자를 이용한 멤버변수 초기화
        int area2 = tr.findArea();
        System.out.println("삼각형의 면적 : " + area2);
    }
}
```

실습1 – 클래스 디자인

- 직사각형을 나타내는 Rectangle클래스 디자인
  - 멤버변수(필드) – 가로, 세로
  - 넓이를 구하는 기능
  - 둘레를 구하는 기능
  - => 생성자를 이용해서 가로, 세로 초기화

```
사각형의 가로, 세로를 입력하세요
10
5
사각형의 넓이 : 50
사각형의 둘레 : 30
```

실습2

- 1. 명함정보를 지닐 수 있는 클래스 정의
  - NameCard Class
    - 멤버변수(필드) : 이름, 전화번호, 주소, 직급
    - 생성자- 멤버변수(필드) 초기화
    - 메서드
      - 필드내용을 화면에 출력하는 메서드
  - 메인 클래스의 main()메서드에서
    - 사용자에게 입력 받고
    - NameCard Class의 화면 출력 메서드 호출하여 화면에 출력하기

```
이름, 전화번호, 회사주소, 직급을 입력하세요
홍길동
710-1234
서초구 서초동 100
사장
==========================================
이름 : 홍길동
전화번호 : 710-1234
회사주소 : 서초구 서초동 100
직급 : 사장
```

실습2 – 계속

- 2. 시, 분, 초 정보를 지닐 수 있는 Time 클래스 정의
  - Time 클래스
    - 멤버변수 – 시, 분, 초
    - 메서드
      - 1) 멤버변수가 지니고 있는 데이터를 출력하는 메서드
        - 출력방식 – 10시 20분 30초
      - 2) 초단위로 계산한 값을 반환하는 메서드 => hour*60*60 + min*60 + sec
    - 생성자
      - 오버로딩(매개변수의 개수가 다른 생성자 3개)
        - 매개변수 : (시, 분, 초), (시, 분), (시)
  - 메인 메서드에서
    - 사용자로부터 시, 분, 초 입력 받기
    - 시, 분, 초를 출력하는 메서드를 호출하여 출력하기
    - 초단위로 계산한 메서드를 호출하여 초를 리턴 받아서 화면에 출력하기

실습2 – 계속

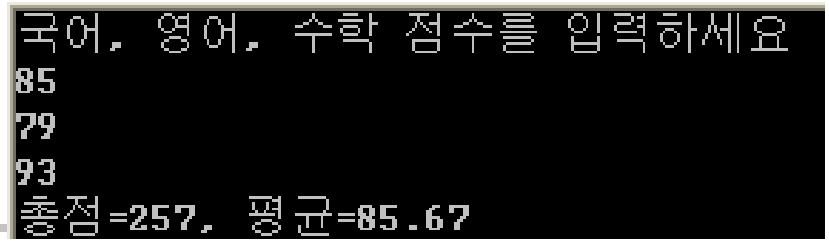
```
시, 분, 초를 입력하세요
10
20
30
================================================
10시    20분    30초
초로 환산하면 37230초
10시    20분    0초
초로 환산하면 37200초
10시    0분     0초
초로 환산하면 36000초
```

실습3

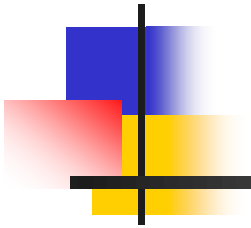
```
국어, 영어, 수학 점수를 입력하세요
85
79
93
총점=257, 평균=85.67
```

- 국어, 영어, 수학 점수를 입력 받아 총점과 평균을 구하는 성적처리 프로그램 만들기
  - 1. 기존 방식대로 처리해보기
    - main() 메서드 내에서 총점, 평균을 구하는 방법
  - 2. 총점, 평균을 메서드로 만들어서 처리하는 방법
    - main() 메서드가 속한 클래스에 총점 구하는 메서드, 평균 구하는 메서드 만들기

실습3



- 3. 성적 클래스 만들어서 처리하기
  - 성적(Score) 클래스 만들기
    - 멤버변수(필드) : 국어, 영어, 수학
    - 메서드
      - 총점 구하는 메서드,
      - 평균 구하는 메서드(매개변수는 없고, 반환값은 있는 메서드로 만들기: 매개변수 대신 멤버변수를 이용해서 계산)
    - 생성자 : 국어, 영어 ,수학 점수 넣기
  - 메인 메서드에서
    - 사용자로부터 국,영, 수 점수 입력 받기
    - 총점, 평균을 구하는 메서드를 호출하여 총점과 평균을 구한 후
    - 결과를 화면에 출력하기



접근 제한자

접근 제한자(접근 제어자)

- 멤버나 클래스에 사용되어, 해당하는 멤버나 클래스를 **외부에서 접근하지 못하도록** 제한하는 역할
- 은닉성 – 객체는 필요한 것만 외부에 노출하고 그 외의 모든 것은 숨기게 됨
- 클래스 내부의 멤버(member)를 **노출하거나 숨길 때** 접근 제한자 사용

• 접근 제어자가 사용될 수 있는 곳 – 클래스, 멤버변수, 메서드, 생성자

1) private – 같은 클래스 내에서만 접근 가능
2) default(생략형) – 같은 패키지 안에 있는 클래스들끼리만 접근 가능
3) protected – 같은 패키지는 물론 다른 패키지일지라도
   상속 관계가 있으면 접근 가능
   – 같은 패키지 내에서, 그리고 다른 패키지의 자식 클래스에서
   접근이 가능함
4) public : 어디서나 접근 가능

접근제어자의 관계

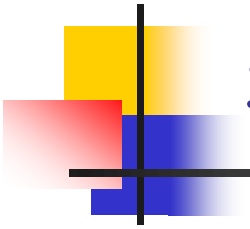
	같은 클래스	같은 패키지	자식 클래스	전체
지시자	클래스 내부	동일 패키지	상속받은 클래스	이외의 영역
private	●	×	×	×
default	●	●	×	×
protected	●	●	●	×
public	●	●	●	●

```
public > protected > default > private
```

접근 범위가 넓은 쪽에서 좁은 쪽의 순으로 왼쪽부터 나열

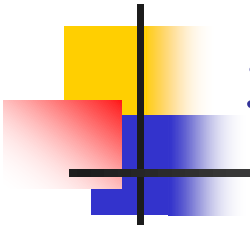
접근 제한자

대상	사용 가능한 접근 제한자
클래스	public, (default)
메서드	public, protected, (default), private
멤버변수	
지역변수	없음



접근 제한자

- 접근 제한자를 사용하는 이유
  - 클래스의 내부에 선언된 데이터를 보호하기 위해서
    - 데이터가 유효한 값을 유지하도록, 비밀번호와 같은 데이터를 외부에서 함부로 변경하지 못하도록 외부로부터의 접근을 제한
  - 외부에는 불필요한, 내부적으로만 사용되는 부분을 감추기 위해서
    - 클래스 내에서만 사용되는, 내부 작업을 위해 임시로 사용되는 멤버변수나 부분작업을 처리하기 위한 메서드 등의 멤버들을 클래스 내부에 감추기 위해서
  - 외부에서 접근할 필요가 없는 멤버들을 private으로 지정하여 외부에 노출시키지 않음

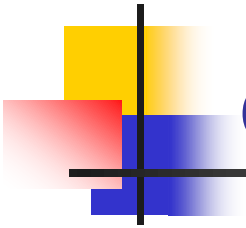


접근 제한자

- 클래스 – 일반적으로 멤버변수는 외부에 대해 숨기고(private), 대부분의 메서드는 노출함(public)
  - 멤버변수 – private : 클래스 외부에서 접근불가, 내부에서만 접근 가능
  - 메서드 – public : 외부에서도 접근 가능
- 클래스의 멤버변수에 값을 입력하고 싶을 때 메서드에 매개변수를 넘기는 방법을 이용
  - 멤버변수는 private으로 지정하고 그 변수를 변경시키거나 값을 가져오는 메서드를 public으로 선언해서 외부에서도 접근 가능하게 함

private 예제

```
class PrivateTest1
{
    public static void main(String[] args)
    {
        AAA obj = new AAA();
        obj.display();
        System.out.println("x=" + obj.x); //에러 x has private access in OtherClass1
    }
}
class AAA
{
    private int x=10;

    public void display()
    {
        System.out.println("x=" + x);
    }
}
```



default 예제

```
class OtherClass2
{
    int x=10;
}
public class ProtectedTest
{
    public static void main(String[] args)
    {
        OtherClass2 obj = new OtherClass2();
        System.out.println("x=" + obj.x);
    }
}
```

- 접근 제한자를 사용하지 않는 형태가 default
- 동일 파일, 동일 폴더(패키지) 내에서 사용 가능

protected 예제

```
class OtherClass2
{
    protected int x=10;
}
public class ProtectedTest
{
    public static void main(String[] args)
    {
        OtherClass2 obj = new OtherClass2();
        System.out.println("x=" + obj.x);
    }
}
```

```
class AAA
{
    protected int num;
    .....
}
class BBB extends AAA
{
    public init(int n)
    {
        num = n; //상속된 변수 num에 접근
    }
}
```

- 동일한 파일내에서는 사용 가능
- 서로 다른 파일로 작성되고 동일한 패키지(폴더) 내에만 있어도 사용 가능
- 다른 패키지(폴더)일지라도 상속 관계가 있으면 접근 가능

default 클래스

- default 클래스-동일한 패키지내에 정의된 클래스에 의해서만 인스턴스 생성이 가능

```
package apple;
class AAA    // default 클래스 선언
{
     . . . .
}

package peal;
class BBB    // default 클래스 선언
{
     public void make()
     {
          apple.AAA inst=new apple.AAA();
          . . . .
     }
     . . . .
}
```

인스턴스 생성불가!
AAA와 BBB의 패키지가 다르므로!

public클래스

- public 클래스 – 어디서나 인스턴스 생성이 가능
  - 하나의 소스파일에 하나의 클래스만 public으로 선언가능
  - public 클래스 이름과 소스파일 이름은 일치해야 함.

```
package apple;
public class AAA     // public 클래스 선언
{
     . . . .
}

package peal;
public class BBB     // public 클래스 선언
{
     public void make()
     {
          apple.AAA inst=new apple.AAA();
          . . . .
     }
     . . . .
}
```

AAA는 public 클래스이므로
어디서든 인스턴스 생성가능

생성자는 public인데, 클래스는 default?

```
public class AAA
{
    AAA(){…}
    . . . .
}
```

클래스는 public으로 선언되어서 파일을 대표하는 상황!
그럼에도불구하고 생성자가 default로 선언되어서 동일패키지내에서만 인스턴스생성을 허용하는상황!

```
class BBB
{
    public BBB(){…}
    . . . .
}
```

생성자가 public임에도 클래스가 default로 선언되어서
동일패키지 내에서만 인스턴스 생성이 허용되는 상황!

아구가 맞지 않는 상황들 => 주의

디폴트 생성자

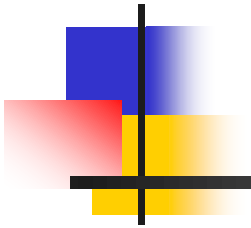
- 디폴트 생성자의 접근제어 지시자는 클래스의 선언형태에 따라서 결정됨

```
public class AAA
{
    public AAA() {…}
    . . . .
}
```

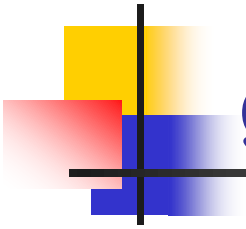
public 클래스에 디폴트로 삽입되는 생성자

```
class BBB
{
    BBB() {…}
    . . . .
}
```

default클래스에 디폴트로 삽입되는 생성자



getter/setter



getter/setter

- 클래스
  - 데이터에 해당하는 필드
    - private으로 선언
  - 기능에 해당하는 메서드
    - public으로 선언
    - 멤버변수의 값을 가져오거나 변경하기 위해서 get~(), set~() 메서드 만들어 사용
      - getter/setter
    - 멤버변수 하나당 한 쌍의 get~(), set~() 메서드 필요

멤버 변수(private)의 값 조정

```
m의 키는 180cm
m의 나이는 20살
```

- 클래스 내부의 멤버 변수(private)의 값 조정하기
  - public 메서드를 통해 멤버 변수의 값 조정

```
class Man
{
	private int age=0 ;
	int height ;

	public void setAge(int a)
	{
		age = a;
	}
	public int getAge()
	{
		return age;
	}
}
public class ManTest
{
	public static void main(String[] args)
	{
		Man m = new Man();
		m.height = 180 ;
		m.setAge(20);
		System.out.println("m의 키는 " +m.height + "cm");
		System.out.println("m의 나이는 " + m.getAge() +"살");
	}
}
```

예제

```
m의 나이는 27살이다.
```

```
class Man
{
    private int age=0 ;
    public void setAge(int age)
    {
        if(age<1){
          this.age = 1;
        }
        else{
          this.age = age;
        }
    }
    public int getAge()
    {
          return age;
    }
}
public class ManTest2
{
    public static void main(String[] args)
    {
          Man m = new Man();
          m.setAge(27);
          System.out.println("m의 나이는 " + m.getAge()  + "살이다.");
    }
}
```

예제-시간을 표시하기 위한 Time 클래스

```
시간을 잘못 입력했어요!
0시
14시
```

```
class TimeTest
{
    public static void main(String[] args)
    {
        Time t1 = new Time();

        t1.setHour(27);
        int hour=t1.getHour();
        System.out.println( hour + "시");

        t1.setHour(14);
        System.out.println(t1.getHour() +"시");
    }
}
class Time
{
    //1. 멤버필드
    private int hour;
    private int min;
    private int sec;
```

예제-시간을 표시하기 위한 Time 클래스

```
//2. getter/setter
public void setHour(int hour){
        if (hour<0 || hour>23)
        {
                //System.out.println("시간을 잘못 입력했어요!");
                return;
        }
        this.hour=hour;
}
public void setMin(int min){
        if (min<0 || min>59){
                //System.out.println("시간을 잘못 입력했어요!");
                return;
        }
        this.min=min;
}
public void setSec(int sec){
        if (sec<0 || sec>59){
                //System.out.println("시간을 잘못 입력했어요!");
                return;
        }
        this.sec=sec;
}
```

예제-시간을 표시하기 위한 Time 클래스

```
    public int getHour()
    {
            return hour;
    }
    public int getMin()
    {
            return min;
    }
    public int getSec()
    {
            return sec;
    }
}
```

예제

```
1900년 이상만 지정이 가능합니다.
BirthYear : -1, Name : 홍길동
```

```
BirthYear : 1988, Name : 홍길동
```

```
class Employee
{
	private int birthYear=-1;
	private String name;

	public int getBirthYear()
	{
		return this.birthYear;
	}
	public void setBirthYear(int birthYear)
	{
		//입력 범위가 아닐 경우 예외 발생
		//별도 로직 처리가 가능하다.
		if (birthYear < 1900)
		{
			System.out.println("1900년 이상만 지정이 가능합니다.");
			return;
		}
		this.birthYear = birthYear;
	 }
```

예제3

```
    public String getName()
    {
        return this.name;
    }
    public void setName(String name)
    {
        this.name = name;
    }
}//
```

예제3

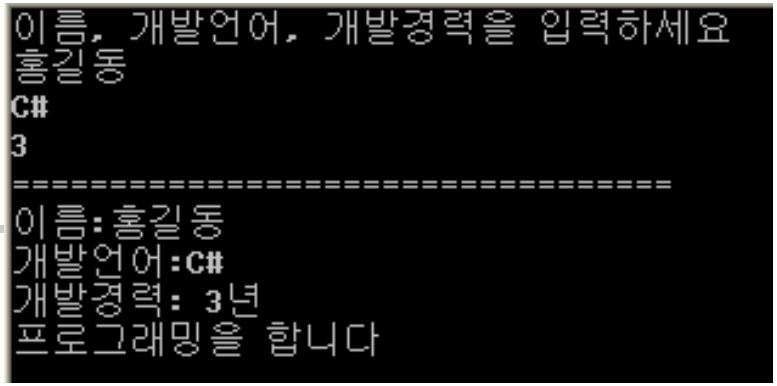
```
public class EmployeeTest2
{
    public static void main(String[] args)
    {
        Employee emp = new Employee();
        emp.setBirthYear(1888);          //예외 발생 : 1900 이상만 가능
        emp.setName("홍길동");

        System.out.println("BirthYear : " + emp.getBirthYear() + ", Name : " +
    emp.getName());
    }
} //
```

실습

```
이름, 학번을 입력하세요
홍길동
201216037
==================================
이름:홍길동
나이:201216037
```

- Student 클래스
  - 멤버변수 – 이름(name), 학번(idNo)
  - getter/setter 만들기
- main()에서 setter로 멤버변수에 값을 넣어주고, getter로 값을 읽어서 화면에 출력하기

실습



- Programmer 클래스
  - 필드 : 이름(name), 개발언어(language), 개발경력(career)
  - getter/setter 만들기
  - 메서드 : 하는 일을 출력 - work()
    - "프로그래밍을 합니다."
- main()에서는
  - (1) 이름, 개발언어, 개발경력을 입력 받아
    - Programmer 객체 생성 후 값을 넣어준 후, 화면 출력 (getter/setter 이용)
    - 하는 일 메서드 호출

예제 - 삼각형 면적 구하기 **getter/setter** 이용

```
import java.util.*;
class Triangle{
    //1. 멤버변수
    private int width;  //밑변
    private int height;  //높이

    //2. 생성자
    Triangle(int w, int h){
            width=w;
            height=h;
    }

    //3. getter/setter
    public int getWidth(){  //읽기
            return width;
    }
    public void setWidth(int w){  //쓰기
            width=w;
    }

    public int getHeight(){
            return height;
    }
    public void setHeight(int h){
            height=h;
    }
```

```
    //4. 메서드 - 삼각형 면적 구하는 메서드
    public int findArea(){
        int area = width*height/2;
        return area;
    }
}
class TriangleTest4 {
    public static void main(String[] args)    {
        Scanner sc = new Scanner(System.in);
        System.out.println("삼각형의 밑변, 높이 입력!");
        int w = sc.nextInt();
        int h = sc.nextInt();

        //생성자를 이용한 초기화
        Triangle tr = new Triangle(w, h);
        int area = tr.findArea();
        System.out.println("삼각형의 면적 : " + area);

        //getter/setter 이용
        w=10;
        h=20;
        tr.setWidth(w);
        tr.setHeight(h);
        area = tr.findArea();
        System.out.println("밑변 : " + tr.getWidth()+", 높이 : " + tr.getHeight());
        System.out.println("면적 : " + area);
    } }
```

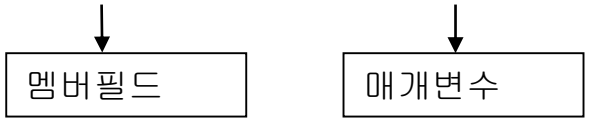
자기 참조 – this 키워드

- this – 참조변수로, 인스턴스 자신을 가리킴
- 참조변수를 통해 인스턴스의 멤버에 접근할 수 있는 것처럼, this로 인스턴스 변수에 접근할 수 있음
- this를 사용할 수 있는 것은 인스턴스 멤버뿐임
- 인스턴스의 주소가 저장되어 있음

- 객체를 메모리에 load했을 때 메모리에 load된 자기자신을 나타낼 때 사용
- 클래스 내부에서 자기 자신의 클래스를 지칭하는 객체로 이용됨
- 클래스 내에서 클래스가 가지고 있는 멤버필드 또는 멤버 메서드를 직접 참조할 수 있는 자신의 참조변수
  - 자신의 멤버를 가리키는 this (this.멤버)
- **클래스를 디자인할 때 사용,** 객체 생성 후 사용하는 것이 아님
  - 클래스 안에서만 사용가능, 클래스 밖에서는 사용할 수 없음
  - static 메서드에서는 사용할 수 없음
- 객체를 생성하기 전 단계의 그 주소를 this라 칭하고, 할당되는 순간 this에게 할당된 메모리의 참조값을 넘겨줌

자기 참조 – this 키워드

```
public class Account
{
    private int balance ;                  // 잔 액

    public Account(int balance)     // 생성자
    {
        this.balance = balance;
    }
}
```



- this - 인스턴스 자신을 가리키는 참조변수,
  인스턴스의 주소가 저장되어 있음

예제

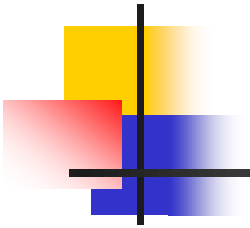
- 이름, 나이, 전화번호를 갖는 Person 클래스 작성
  - 데이터 – 이름, 나이, 전화번호
  - 기능
    - 설정되어 있는 이름, 나이, 전화번호를 출력하는 기능
  - 초기화
    - 이름, 나이, 전화번호를 설정하는 생성자

Person 클래스

```
class Person
{
	//1. 멤버필드
	private String name;
	private int age;
	private String phone;

	//2. 생성자
	public Person(String name, int age, String phone)
	{
		this.name = name;
		this.age = age;
		this.phone = phone;
	}

	//3. 멤버메서드
	public void Display()
	{
		System.out.println("이름 : " + name);
		System.out.println("전화번호 : " + phone);
		System.out.println("나이 : " + age);
	}
};
```

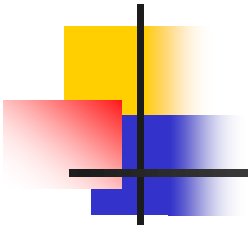
Person 클래스-계속

```
import java.util.*;
class PersonTest
{
	public static void main(String[] args)
	{
		System.out.println("이름, 전화번호, 나이를 입력하세요");
		Scanner sc = new Scanner(System.in);
		String name = sc.nextLine();
		String phone = sc.nextLine();
		int age = sc.nextInt();

		System.out.println("=========================================");

		Person p = new Person(name, age, phone);
		p.Display();
	}
};
```



변수

- 변수
  - **멤버변수(필드, 인스턴스 변수)**
    - 클래스에서 선언된 변수
    - 클래스의 멤버역할을 하는 member field
    - 클래스 내의 여러 메서드에서 사용 가능, 클래스 외부에서도 접근 가능하게 할 수 있음
    - 0(숫자필드), false(논리형), null(참조형)값으로 초기화함
  - **지역변수**(Local variables)
    - 메서드 내부에서만 사용 가능한 지역변수, 메서드 내에서 선언되는 변수
      - 메서드가 시작될 때 생성
      - 메서드를 빠져나갈 때 사라짐
  - ※ **블럭변수** – 메서드내의 또 다른 블록(if, for등)내에서 선언된 변수

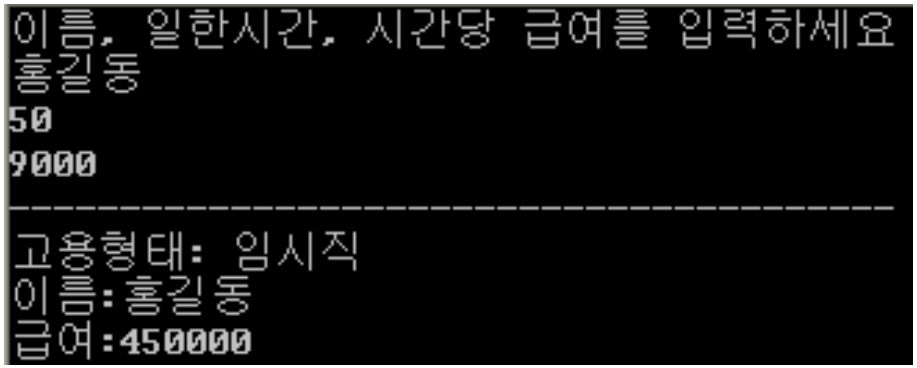


객체 LifeCycle

- 객체 생성 시기
  - new 연산자를 이용한 메모리를 할당하고 객체를 생성
  - 생성자를 이용해 메모리 상의 객체 초기화
- 객체 사용 시기
  - 객체에 대한 메서드, 멤버 호출 / 접근
- 객체 소멸 시기
  - 객체에게 할당된 메모리는 실행 환경으로 회수
  - 메모리 해제
  - 소멸 시기가 비결정적임
  - Garbage Collector (GC)가 객체 소멸을 관장함

실습 – 급여관리

- 임시직 클래스 정의 (Temporary)
  - 멤버변수 : 이름(name), 일한시간(time), 시간당 급여(pay)
  - 생성자, getter/setter
  - 메서드 : 급여를 계산하는 메서드
    - => 일한시간 * 시간당 급여
- main 에서
  - 사용자로부터 이름, 일한 시간, 시간당 급여를 입력 받아서
  - 생성자로 초기화
  - 급여계산 메서드 호출하여 급여를 계산한 후
  - 결과 출력



과제1 -가위바위보 게임

- 컴퓨터가 랜덤하게 낸 가위바위보와 사용자가 입력한 가위바위보를 비교하여 누가 이겼는지 결과를 알려주기
  - 숫자를 랜덤하게 뽑아서
    - 0 => 가위, 1=>바위, 2=>보
  - 클래스 추상화
    - 속성(변수)
      - result. 결과값(이기다, 지다, 비기다)
      - user. 사용자가 선택한 값
      - com. 컴퓨터가 선택한 랜덤값
    - 기능(메서드)
      - 비교하다(사용자값과 컴퓨터값을 비교)

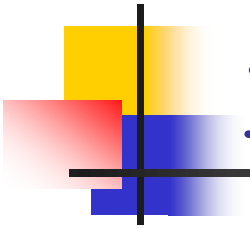
- 사용자가 이기는 경우의 수

사용자 (a)			컴퓨터 (b)
이김	짐	비김	
0 (가위)	1	2	2 (보)
1 (바위)	2	0	0 (가위)
2 (보)	0	1	1 (바위)
규칙 : (a-b+3) % 3 => 1: 이김, 2:짐, 0:비김			

```
가위(0), 바위(1), 보(2), Q(Quit)를 입력해주세요
2
사용자 =보
컴퓨터 = 가위
결과 = 사용자가 졌습니다
```

과제1 – 계속

- 가위바위보 클래스 만들기
  - 멤버변수
    - user(가위, 바위, 보), com(가위, 바위, 보),
    - result(이겼습니다, 졌습니다, 비겼습니다)
  - 메서드
    - 1. 비교하기 메서드
      - 사용자 입력값(iuser =>0, 1, 2)과 컴퓨터랜덤값(icom=> 0, 1, 2)을 인자로 받기
      - (iuser – icom + 3) % 3 에 의해
        - 결과가 1이면 result(멤버변수)는 “이겼다”
        - 결과가 2이면 result(멤버변수)는 “졌다”
        - 결과가 0이면 result(멤버변수)는 “비겼다”
      - 멤버변수 user(사용자 선택값)과 com(컴퓨터 랜덤값) 셋팅
        - 0 => 가위, 1=> 바위, 2=>보
    - 2. 각각의 멤버변수를 리턴해 주는 getter 3개
    - 3. 0, 1, 2 숫자를 가위, 바위, 보로 변환해주는 private 메서드



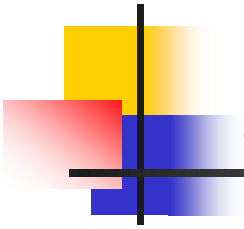
과제1 – 계속

- 메인클래스의 메인함수에서
  - 1. 사용자에게 0, 1, 2 중 입력 받고,
  - 2. 컴퓨터가 랜덤하게 선택한 값 (0<= x < 3)
    Math.random()
  - 3. 가위바위보 클래스의 비교하기 메서드 호출하여 비교
  - 4. 결과값과 사용자, 컴퓨터가 낸 가위, 바위, 보 출력

과제2 – 전화번호 관리 프로그램 작성하기

- [1단계]
- 전화번호 정보 클래스 정의(PhoneInfo)
  - 데이터 – 이름, 전화번호, 생년월일
  - 기능 – 데이터를 출력하는 메서드
    - 단, 생년월일 정보는 저장을 할 수도, 저장하지 않을 수도 있도록 생성자를 정의할 것(매개변수가 3개인 생성자와 2개인 생성자를 정의)
    - 생년월일 정보를 몰라도 저장되도록
- main() 메소드에서 객체 생성시 임의의 값으로 초기화하고, 출력 메서드를 호출해서 화면 출력하기

```
name: 홍길동
phone: 010-100-2000
birth: 92-01-17

name: 김연아
phone: 010-300-4000
```



과제2 – 전화번호 관리 프로그램 작성하기

- [2단계]
  - 사용자로부터 데이터를 입력 받아서 PhoneInfo클래스의 객체를 생성하기
  - 반복문을 이용해서 프로그램의 흐름이 유지되도록 한다.
  - 사용자가 종료를 선택하지 않으면 다음의 과정이 반복적으로 이루어지도록 한다.
    - 1. 키보드로부터 데이터 입력
    - 2. 입력 받은 데이터로 PhoneInfo 클래스의 객체 생성
    - 3. 생성된 객체의 화면출력 메서드 호출
  - 프로그램의 흐름을 계속해서 이어갈지, 아니면 종료할지 사용자가 선택하게 한다.

과제2 – 전화번호 관리 프로그램 작성하기

```
선택하세요...
1. 데이터 입력
2. 프로그램 종료
선택: 1
이름: 홍길동
전화번호: 010-100-2000
생년월일: 92-04-09

=======입력된 정보 출력========
name: 홍길동
phone: 010-100-2000
birth: 92-04-09

선택하세요...
1. 데이터 입력
2. 프로그램 종료
선택: 2
프로그램을 종료합니다.
```

메서드 테스트

MemberGrade 클래스

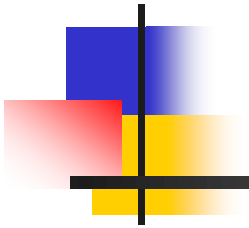
멤버변수 : 거래횟수, 연체 횟수
생성자
메서드

- 1. 메서드 만들기
- 거래 횟수와 연체 회수에 따른 회원등급 구하는 메서드 만들기
  - 거래 횟수가 20회 이상이고, 연체 회수가 0회 이하인 회원은 "우수회원", 나머지 회원은 "일반회원"

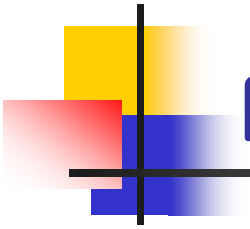
- 2. main() 에서 사용자로부터 거래 회수와 연체 회수를 입력받아서 회원등급을 구하는 메서드를 호출한 후 결과를 출력한다.

```
거래회수와 연체 회수를 입력하세요
30
0
거래횟수 : 30, 연체횟수 : 0=> 회원등급 : 우수회원
```

```
거래회수와 연체 회수를 입력하세요
20
5
거래횟수 : 20, 연체횟수 : 5=> 회원등급 : 일반회원
```



참고 – 과일장사 클래스 만들기

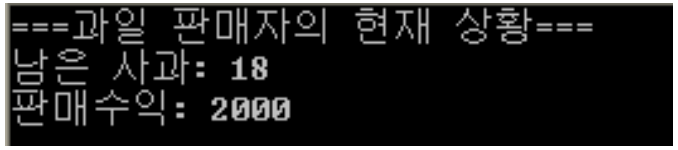


FruitSeller 클래스 정의

- 예) 나는 과일장사에게 2000원을 주고 두 개의 사과를 구매했다.
  - 필요한 클래스 정의하기
  - FruitSeller 클래스에 한 가지 기능 추가
    - 오늘 2000원 벌었고, 남은 사과는 18개다.

FruitSeller 클래스

```
class FruitSeller
{
    final int APPLE_PRICE=1000; //사과 가격
    int numOfApple=20;
    int myMoney=0;

    public int saleApple(int money)
    {
        int num=money/APPLE_PRICE;
        numOfApple -= num;
        myMoney += money;
        return num;
    }

    public void showSaleResult()  //추가된 메서드
    {
        System.out.println("남은 사과: "+ numOfApple);
        System.out.println("판매수익: "+ myMoney);
    }
}
```



예제 완성하기

```
class FruitSalesMain
{
    public static void main(String[] args)
    {
        FruitSeller seller = new FruitSeller();
        seller.saleApple(2000); //2000원어치 사과 판매

        System.out.println("===과일 판매자의 현재 상황===");
        seller.showSaleResult();
    }
}
```

예 – 생성자

- 두 명의 과일 장사가 있고, 이들의 판매내용
  - 과일장사1 : 보유하고 있는 사과의 개수는 30개, 개당 가격은 1500원
  - 과일장사2 : 보유하고 있는 사과의 개수는 20개, 개당 가격은 1000원
- 나는 과일장사1 에게 4500원어치 사과를 구매했고, 과일장사2 에게 2000원어치 사과를 구매했다.
  - 두 개의 과일장사 객체 생성
  - 과일장사의 사과 보유수와 개당 가격이 다르므로, 변수의 초기 값도 달라져야 함
  - 클래스를 정의하면서 변수 값을 초기화할 수 없음
  - 객체 생성 후, 멤버변수를 각각 초기화하자

예

초기화가 이뤄지지 않은 final 변수는 한 번의 초기화 기회를 갖는다

```
class FruitSeller
{
    int APPLE_PRICE; //사과 가격  //final 선언 사라짐=> 메서드내에서 값을 변경하기 위해
    int numOfApple;
    int myMoney;

    public void initMembers(int money, int aNum, int price)
    {
        myMoney=money;
        numOfApple=aNum;
        APPLE_PRICE=price;
    }
…
}
```

예

```
===과일 판매자1의 현재 상황===
남은 사과: 27
판매수익: 4500

===과일 판매자2의 현재 상황===
남은 사과: 18
판매수익: 2000
```

```
class FruitSalesMain2
{
	public static void main(String[] args)
	{
		FruitSeller seller1 = new FruitSeller();
		seller1.initMembers(0, 30,1500); //두 줄에 걸쳐서 문장을 구성해야 하나의 인스턴스 생성이 완료된다는 문제점

		FruitSeller seller2 = new FruitSeller();
		seller2.initMembers(0, 20,1000);

		seller1.saleApple(4500);
		seller2.saleApple(2000);

		System.out.println("===과일 판매자1의 현재 상황===");
		seller1.showSaleResult();

		System.out.println("₩n===과일 판매자2의 현재 상황===");
		seller2.showSaleResult();
	}
}
```

예제-생성자 이용

```
===과일 판매자1의 현재 상황===
남은 사과: 27
판매수익: 4500

===과일 판매자2의 현재 상황===
남은 사과: 18
판매수익: 2000
```

```
class FruitSeller
{
	final int APPLE_PRICE; //사과 가격
	int numOfApple;
	int myMoney;

	public FruitSeller(int money, int aNum, int price)
	{
		myMoney=money;
		numOfApple=aNum;
		APPLE_PRICE=price;
	}

	public int saleApple(int money)
	{
		int num=money/APPLE_PRICE;
		numOfApple -= num;
		myMoney += money;
		return num;
	}

	public void showSaleResult()  //추가된 메서드
	{
		System.out.println("남은 사과: "+ numOfApple);
		System.out.println("판매수익: "+ myMoney);
	}
}
```

예제

```
class FruitSalesMain3
{
	public static void Main()
	{
		FruitSeller seller1 = new FruitSeller(0,30,1500);
		FruitSeller seller2 = new FruitSeller(0,20,1000);

		seller1.saleApple(4500);
		seller2.saleApple(2000);

		System.out.println("===과일 판매자1의 현재 상황===");
		seller1.showSaleResult();

		System.out.println("₩n===과일 판매자2의 현재 상황===");
		seller2.showSaleResult();
	}
}
```

정보 은닉

```
===과일 판매자의 현재 상황===
남은 사과: 10
판매수익: 500

===과일 구매자의 현재 상황===
현재 잔액: 9500
사과 개수: 20
```

외부에서 멤버변수에 직접 접근이 가능했기 때문에 문제 발생

```
class FruitSalesMain
{
    public static void main(String[] args)
    {
        FruitSeller seller = new FruitSeller(0,30,1500);
        FruitBuyer buyer = new FruitBuyer(10000);

        seller.myMoney += 500;  //돈 500원 내고,
        buyer.myMoney -= 500;

        seller.numOfApple -= 20;  //사과 20개 가져가기
        buyer.numOfApple += 20;


        System.out.println("===과일 판매자의 현재 상황===");
        seller.showSaleResult();

        System.out.println("₩n===과일 구매자의 현재 상황===");
        buyer.showBuyResult();
    }
}
```