JAVA 5강 배열

강사 박주병

1. 객체

객체라는 것은 세상에 존재하는 모든 것을 말한다. 모니터, 책, 의자 같은 눈으로 보이는 사물뿐 아니라 행복, 기쁨과 같은 감정들, 추상적인 것들 까지 모두 객체이다. 한마디로 인간이 표현할수 있는 모든 것이 객체라고 볼수 있다.

이러한 객체는 데이터와 기능으로 이루어져 있고 이것을 코드로써 표현하면 실세계에 표현되는 모든 것을 소스코드로써 다 표현 가능하다고 보는 것이다.

그림1의 자동차라는 객체는 색상, 기름량, 주행거리 등등 차량의 현재 상태를 나타내는 데이터와 각종 기능들을 가진다. 객체라는 것은 항상 이렇게 데이터 +기능 으로 이루어져 있다고 보고 이러한 방식으로 소스코드를 ksemf면 세상에 있는 모든 것들을 프로그램으로 표현 할수 있다고 보는 것이 객체지향적 프로그래밍이다.

그림 1

가. 클래스와 객체

클래스는 데이터와 기능으로 이루어진 객체를 만들기 위한 설계도이다.

그림 2

그림2를 보면 클래스는 붕어빵틀로 비유해볼수 있다. 붕어빵틀은 붕어빵들이 어떤 크기와 모양이 될지에 대한 틀만 제공을 하고 이러한 틀을 가지고 실제 붕어빵들을 만들어낸 것이 객체이다.

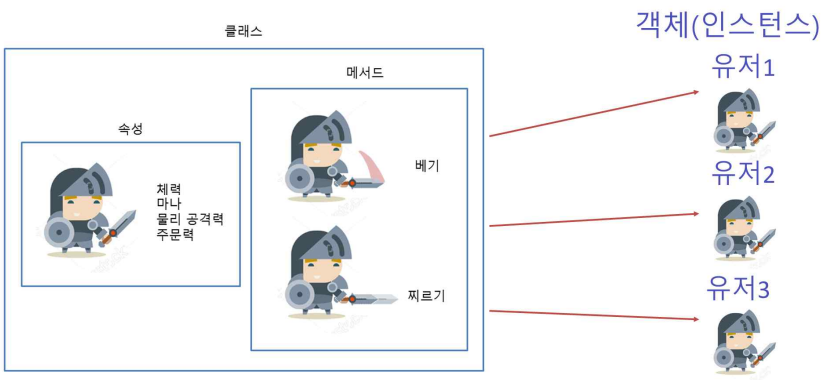


그림 3

이번에는 좀 더 프로그래밍으로 표현을 해보자. 게임 케릭터를 개발한다고 했을 때 케릭터를 만들어낼 설계도가 필요하며 이러한 설계도를 클래스로 표현한다. 클래스는 객체를 만들어 내기 위한 틀이므로 객체를 구성하는 데이터와 기능을 작성해야 한다.

클래스에서 데이터는 속성이라고 부르며 기능을 메서드로 표현한다. 게임 케릭터는 체력,마나와 같은 현재 상태를 나타내는 속성들이 있고 걷기, 뛰기, 찌르기, 베기 같은 기능들을 메서드로 표현한다.
그리고 이렇게 작성해놓은 클래스를 기반으로 속성에 적절한 값을 부여하여 객체들을 만들어 낸다.

```
class Tv
{
    //데이터(멤버변수)
    String color;
    boolean power = false;
    int channel =0;

    //기능(메서드)
    void TurnOn()
    {
        power = true;
    }
    void TurnOff()
    {
        power = false;
    }
    void ChannelUp()
    {
        channel++;
    }
    void ChannelDown()
    {
        channel--;
    }
}
```

이제 실제 코드를 보도록 하자 다음은 TV를 클래스로 표현을 한 코드이다.
class 키워드 옆 클래스 이름을 적어주고 클래스의 범위를 중괄호로 묶어 준다.

참고로 클래스 이름의 첫 글자는 대문자로 적어주었는데 문법적 요소는 아니고 권장사항이다. 하지만 반드시 지킬 것을 권장한다. 차후에 여러 문법들을 배우며 활용하다 보면 대문자를 통해 클래스이름이 명확해지고 가독성이 높아질수 있는 방법이다.

이제 클래스 내부를 보면 데이터(속성)을 나타내는 방법으로는 지금까지 써왔던 변수를 이용해 표현하고 있다.
TV의 색상과 전원연결여부, 현재 채널등의 상태값을 가지는 변수들이다.이렇게 클래스 내부에 있는 변수들을 멤버변수 라고 부른다.

변수는 다음장에서 자세히 다루도록 하고 기능을 나타내는 메서드 부분을 보도록 하자.

객체는 기능을 여러개 가질수 있으므로 클래스를 작성할 때 역시 여러 메서드를 작성할수 있다.

우선 여기서는 클래스를 어떻게 만드며 내부에 멤버변수와 메서드들로 이루어져 있다는것만 확인하고 메서드에 대해서는 조금 뒤에 자세히 다루도록 하자.

나. 클래스간의 관계

이제 설계도를 만들었으면 클래스를 기반으로 객체를 생성하여 코드를 실행하는 것을 해보아야 한다. 클래스만 만들어서는 해당 기능들을 실행할 수가 없다. 예를 들어 TV 설계도만 가지고 TV를 켜서 시청을 할수 있는가?
설계도를 기반으로 실제 TV라는 물건을 생산해서 사용을 해야 하는 것이다. 이렇게 실제 물건을 만드는 것을 객체화(인스턴스화) 한다고 하며 만들어진 TV가 객체인 것이다.

클래스는 일반적으로 java파일 하나당 한 개씩 작성을 한다. 문법적으론 하나의 java파일에 여러개의 클래스 선언이 가능은 하지만 일반적이지 않고 잘쓰이지 않는다.

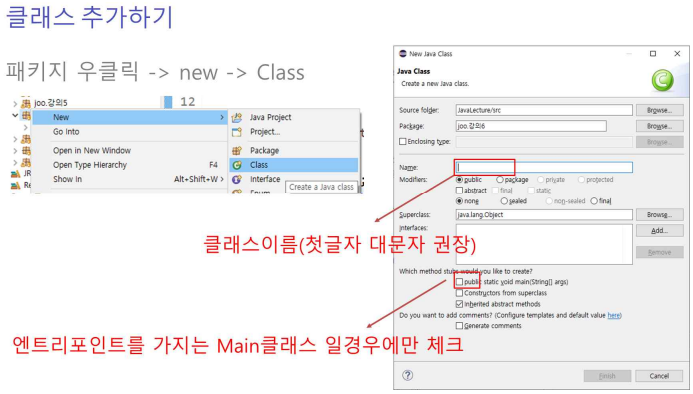


그림 5

그림 6

클래스를 추가 하는 방법은 그림5와 같이 하면 되며 자동으로 만들어 주는 클래스 코드의 내부를 채워 넣으면 된다. 여기서 주의할 점은 java의 파일명과 해당파일 내부에서 작성하는 class의 이름이 대소문자까지 구분하여 정확히 일치해야한다. 그림6을 보면 Tv라고 작성하여 정확히 일치하는 것을 볼 수 있다.

이제 클래스를 기반으로 객체를 생성하고 실행하는 것을 보도록 하자.1강에서 배웠던것처럼 프로그램이 실행되기 위해서는 반드시 main이라는 엔트리포인트가 있어야 하며 이러한 메서드를 가지는 Main이라는 특별한 클래스가 있다고 하였다. 우리는 지금까지 모든 소스코드를 main메서드 내부에 작성을 하였다.
이는 클래스를 사용할때도 마찬가지이다. 아무리 클래스를 위와 같이 선언하여 만들었다 하여도 객체를 생성하고 사용하는 코드가 Main메서드 내부에 없으면 절대 실행되지 않는다. 그저 설계도만 존재하는 상태인 것이다.

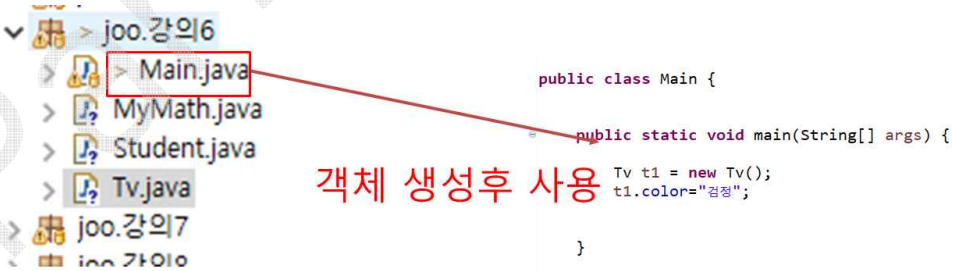


그림 7

그림7을 보면 Main클래스의 내부에 main메서드가 있으며 메서드 내부에 Tv객체를 생성후 사용하는 것을 볼 수 있다. 이렇듯 모든 클래스들은 Main 메서드의 내부에서 객체를 생성 후 사용을 해야 프로그램실행시 실행이 된다.
이제 객체를 생성하는 코드를 자세히 보도록 하자.

```java
public class Main {

    public static void main(String[] args) {

        |
    Tv t1 = new Tv();


    t1.color = "RED";

    t1.TurnOn();
    t1.ChannelUp();


    }

}
```

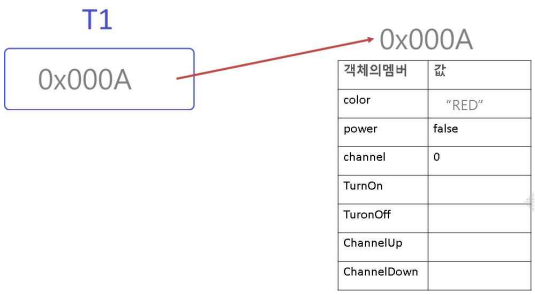
그림 8

그림8의 t1변수를 보면 데이터타입이 지금까지 배운것과 다른 형태의 타입이다. 바로 클래스 이름을 적어주었다. 클래스를 선언해놓았다면 해당 클래스를 기반으로 객체를 생성할텐데 이 객체를 보관해둘 변수의 데이터 타입은 해당 객체의 클래스 이름을 적어주어야 한다.

즉 Tv클래스를 기반으로 만든 객체는 Tv타입의 변수에 넣어두어야 한다.
t1 변수를 초기화 하는 코드를 보면 new 키워드를 사용하고 있다. 객체를 생성하기 위해선 new 키워드를 이용해야하며 오른쪽에 객체를 생성하고자 하는 클래스의 이름과 소괄호를 작성해주어야 한다. 소괄호 부분은 차후에 생성자를 다룰 때 자세히 보도록 하며 지금은 그저 new 클래스이름(); 으로 코드를 작성하면 해당 클래스의 객체가 생성된다는 것을 알아두면 된다.

이제 t1에는 Tv객체가 저장되어 있고 변수명.멤버변수 혹은 변수명.메서드이름(); 등으로 해당 객체의 멤버변수나 메서드 들을 사용 할 수가 있다.

t1은 앞서 객체를 저장한다고 표현하였지만 정확히는 객체 자체를 저장한다기 보다 객체의 메모리 영역의 시작주소를 가지고 있는 것이다.t1과 같이 기본형이 아닌 데이터타입의 변수를 참조타입 변수라고 배웠었다. 참조타입의 변수들은 객체를 저장하기 위한 변수이며 객체의 주소를 가지는 변수이다.
이를 그림으로 표현해보면 다음과같다.



t1 변수는 그저 객체의 시작주소인 0x000A를 가지고 있을뿐이다 메모리의 0x000A주소를 가보면 객체에 대한 정보들이 있는 것이다.

객체가 여러개가 되면 어떻게 될까? 그림10은 Main메서드에서 Tv 객체 2개를 생성하여 각각 t1과 t2 변수에 저장한후 각각의 객체의 멤버변수에 값을 저장하는 코드이다. t1과 t2는 초기화 부분을 보면 new Tv();를 통해 각자 서로 다른 객체를 만들어 내서 저장해둔 것이다 따라서 t1과 t2는 서로 완전히 별개의 객체이고 메모리 구조를 보면 객체의 구성요소인 멤버변수와 메서드들이 완전히 별개의 영역으로 나누어진 것을 볼수 있다. 이렇듯 멤버변수는 객체마다 따로 분리되어 생성되며 각자 값을 따로 가진다.

```java
Tv t1 = new Tv();
Tv t2 = new Tv();

t1.color = "RED";
t2.color = "BLACK";
t1.TurnOn();
t1.ChannelUp();
```

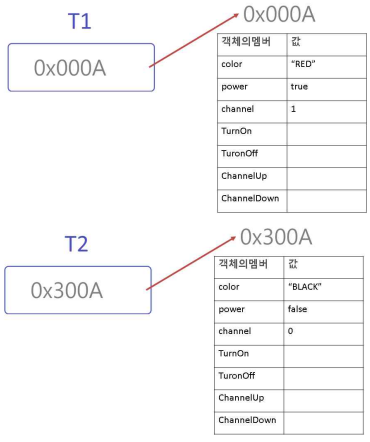


그림 10

예를들어 똑같은 모델의 TV를 각자 만들어 냈다고 하자. TV는 엄연히 2개 이며 각자 색상도 다르며 현재 보고 있는 채널 역시 다를 것이다. 같은 설계도에서 나온 완전히 동일한 모델의 TV라 할지라도 객체는 여러개가 만들어지며 객체마다 고유의 값을 가지는 것이다.

다. 참조변수의 값 교환

참조변수는 기본적으로 실질적인 데이터가 아닌 메모리의 주소를 저장한다. 객체를 저장 할 수 있는 Tv 타입의 변수 T1을 만들고 객체를 생성후 t1변수에 저장한다는 것은 객체의 주소값을 t1에 저장한다는 것이다.

```
Tv t1 = new Tv();
Tv t2 = new Tv();

t1.color = "RED";
t2.color = "BLACK";
t1.TurnOn();
t1.ChannelUp();

t2 = t1;
```

이렇게 참조변수들은 주소값을 가지고 있는데 일반변수와 마찬가지로 참조변수들끼리 주소값을 대입할 수가 있다.
그림11을 보면 t1의 주소값을 t2변수에 대입하는 것을 볼 수 있다. 이렇게 하면 기존의 t1이 가지고 있던 객체의 주소값이 t2에도 저장되게 된다. 결국 t1과 t2가 완전히 동일한 객체를 가리키게 되는 것이다.

그리고 t2가 초기화될 때 생성되었던 객체는 더 이상 어떠한 변수도 가리키고 있지 않은 상태가 되어 객체를 사용할 방법이 없어진다. 그래서 이러한 객체들은 가비지 컬렉션이 수행되며 데이터를 정리 한다.

그림 11

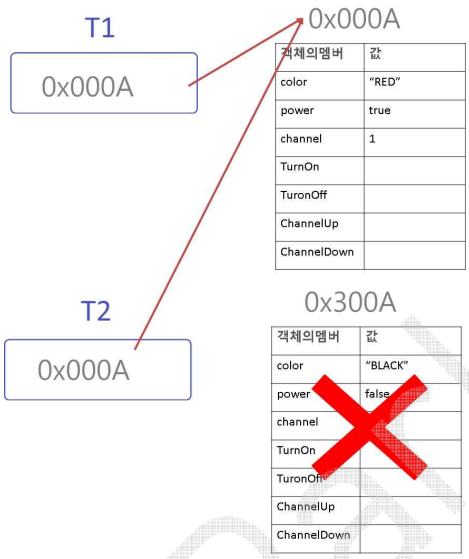


그림12를 보면 t2가 t1과 똑같은 객체의 주소를 가지면서 0x300A 객체는 더 이상 사용 할 수 없게 되어 사라진다.

그림 12

2. 메서드

지금까지는 java 파일을 추가하여 클래스를 선언하고 Main클래스의 Main메서드 내부에서 객체를 생성하고 간단히 멤버변수를 사용하는 것을 봤다. 클래스 내부의 멤버변수에 대해서는 다음장에서 자세히 다루도록 하고 우선 메서드를 살펴보도록 하자.

```
class Tv
{
    String color;
    boolean power = false;
    int channel = 0;

    void turnOn()
    {
        power = true;
    }
    void turnOff()
    {
        power = false;
    }
}
```

그림 13

메서드는 클래스에서 기능을 담당한다. 멤버변수의 값을 변경하며 객체의 상태를 바꾼다던가 객체를 사용하는 외부로부터 값을 넘겨받아 특정 기능을 수행후 값을 다시 되돌려주는등의 기능을 할 수 있다.

그림13의 turnOn() 메서드는 멤버변수power의 값을 true로 변경하는 기능을 한다. 즉 전원을 켜는 기능을 담당하며 그렇기에 전원의 켜짐 유무를 저장하는 power변수의 값을 변경하고 있다.

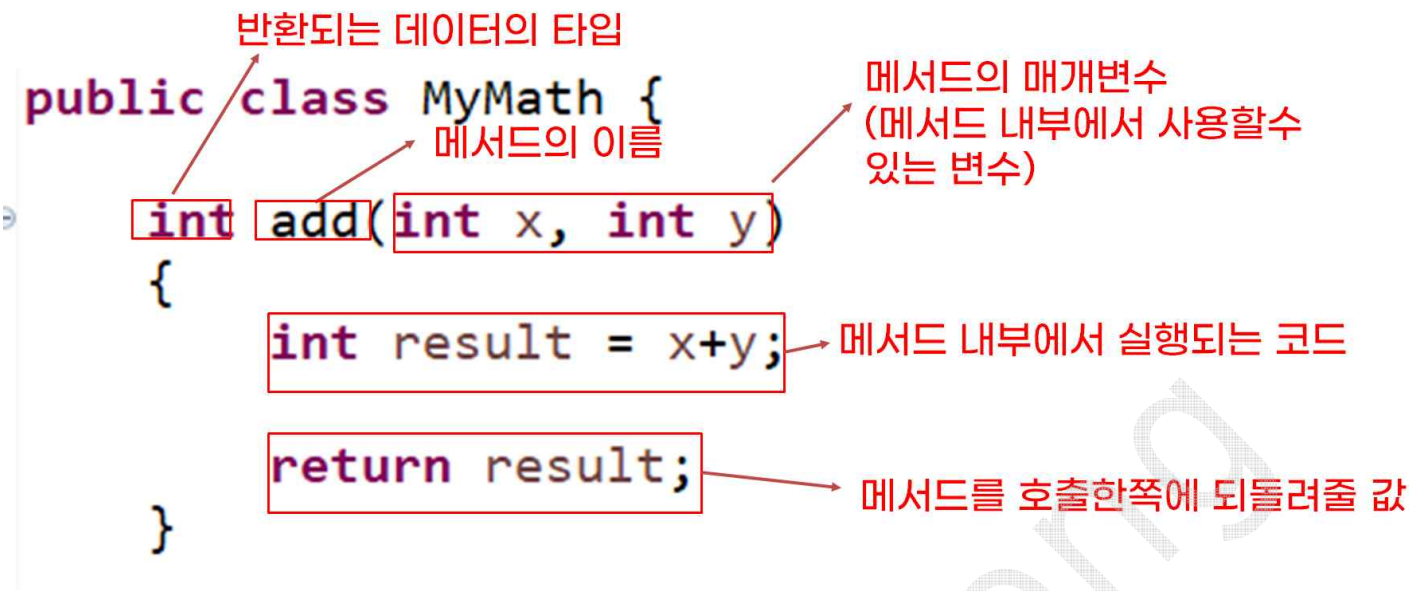
이렇게 Tv 클래스와 turnOn() 메서드를 정의해놓고 그림14처럼 Main클래스에서 Tv 객체를 생성 후 사용 할 수 있다. t1변수를 통해 turnOn() 메서드를 호출하여 사용 할 수 있다.

```
Tv t1 = new Tv();

t1.turnOn();
```

그림 14

가. 메서드의 구조



MyMath 클래스에 add라는 메서드 하나를 선언하였다. 지금부터 메서드의 기본 구조에 대해 알아본다.

int : 가장 처음 나오는 int는 메서드를 호출하는쪽(Main클래스의 main() 메서드 내부)으로 값을 되돌려줄 때 되돌려주는 값의 데이터 타입을 의미한다. 만약 double타입의 값을 되돌려준다면 double를 작성해야 한다.

add : 메서드의 이름이며 관례상 자바에서는 소문자로 시작을 하며 다음 단어부터 첫글자를 대문자로 작성한다.

(int x,int y) : 파라미터, 매개변수 등으로 불리며 메서드이름의 오른쪽에 붙여서 작성하며 메서드를 호출할 때 값을 메서드 내부로 넘겨줄 때 이용한다. 변수의 개수는 제한이 없다.

int result = x+y; : 메서드의 내용부분이며 일반적인 실행 코드들을 작성한다.

return result; : 메서드 내부에서 일정한 처리를 한후 특정값을 메서드를 호출한곳으로 되돌려줄 때 사용한다.
위의 예제에서는 x+y의 결과를 result변수에 저장하였고 해당변수의 값을 메서드를 호출한곳으로 되돌려주고 있다.
return 키워드는 값을 되돌려주고 메서드는 종료 된다. 따라서 return 다음에 작성되는 코드들은 실행 될 수 없다.

나. 메서드의 시그니처

클래스 내부에 메서드를 선언할 때 똑같은 메서드를 중복 정의 할 수 없다. 여기서 똑같은 메서드 라는 것을 판단할 때 무엇이 똑같으면 두 개가 같은 메서드 라고 판단할까? 단순히 이름만 같다고 해서 똑같은 메서드로 인식하는 것이 아니다. 이름뿐 아니라 매개변수의 개수와 매개변수들의 데이터 타입까지 모두 일치해야 같은 메서드이며 시그니처가 같다고 표현한다.
즉 메서드의 시그니처라면 메서드의이름, 매개변수의 개수,매개변수의 타입 이며 하나의 클래스 내부에서 시그니처가 똑같은 메서드를 2개 정의 할수 없다.

```
int add(int x, int y)
{
    int result = x+y;
    return result;
}
```

add(int x,int y) 가 메서드의 시그니처이다.

그림 16

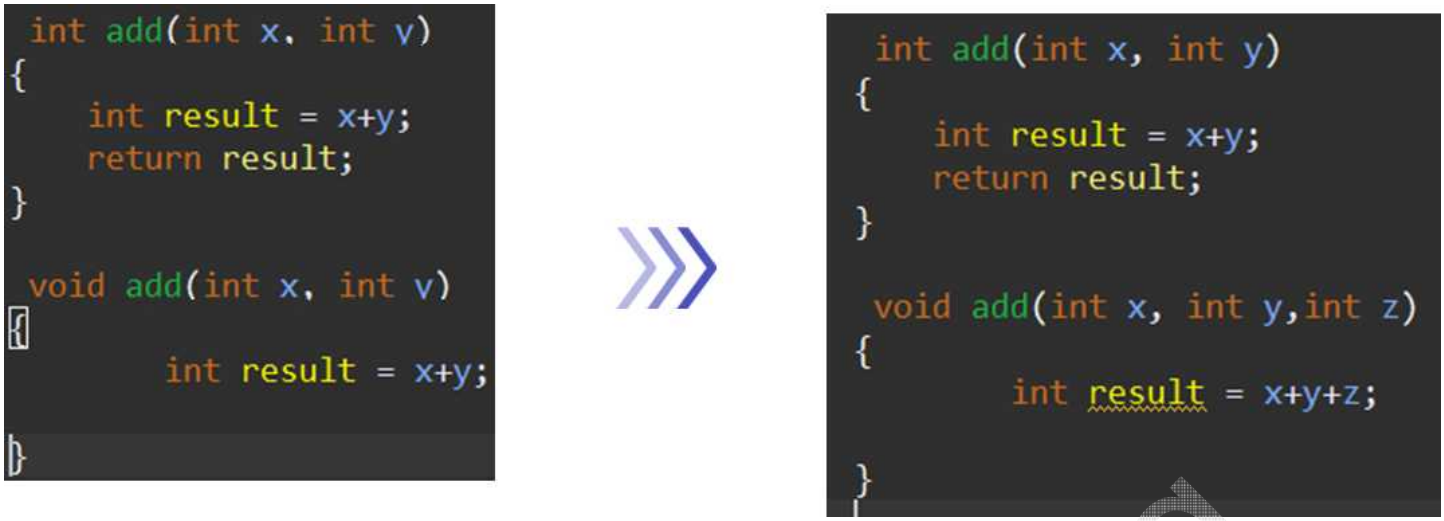


그림 17

그림17의 왼쪽처럼 시그니처가 동일한 메서드는 같은 클래스내에 만들 수 없다. 그러나 오른쪽 같은 경우는 비록 메서드의 이름과 매개변수들의 데이터타입은 동일하지만 매개변수의 개수가 다르므로 2개다 클래스내에 선언할수 있다. 메서드의 활용 부분에서 배울 내용이지만 이렇게 동일한 메서드의 이름으로 여러개의 메서드를 정의 하는 것을 보고 메서드 오버로딩 이라고 부른다.

그리고 당연하지만 클래스가 다르다면 시그니처가 같아도 상관이 없다.

다. 같은 클래스에서 메서드 호출

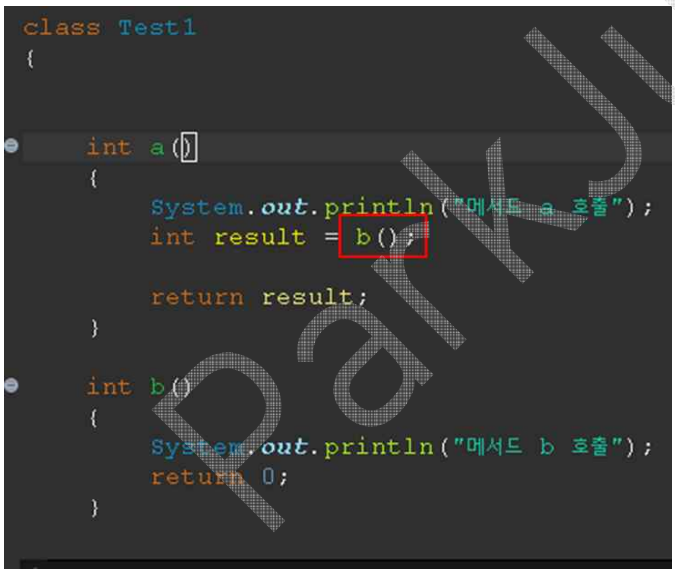


그림 18

그림18을 보면 test1 클래스에는 a() 메서드와 b()메서드가 선언되어 있다. 그러나 a() 메서드 내부를 자세히 보면 b()메서드를 사용하는 것을 볼수 있다.

지금까지 클래스를 선언하고 내부에 메서드를 만들었으면 이를 사용하기 위해선 객체를 생성후 참조변수에 담아두고 참조변수의이름.메서드이름()을 호출하여 메서드를 사용했어야 했다.
하지만 그림18의 예제에서는 b()메서드를 바로 사용하고 있다. 이렇게 같은 클래스 내에서 메서드를 호출 할 때는 객체 생성 없이 호출이 가능하다.

```
class Test1
{
    int a()
    {
        int result = b();

        return result;
    }

    int b()
    {
        int result = c();
        return result;
    }
}

class Test2
{
    int c()
    {
        return 0;
    }

    int d()
    {
        return 0;
    }
}
```

```
int b()
{

    Test2 temp = new Test2();

    int result = temp.c();
    return result;
}
```

그림 19

반면 그림19는 Test1의 b()메서드 내부를 살펴보면 Test2의 c()메서드를 사용하려는 것을 볼 수 있는데 이렇게 클래스가 서로 다르다면 바로 사용 할 수 없다. 오른쪽처럼 Test2의 객체를 생성 후 작성해줘야 한다.

라. void

메서드를 선언할 때 가장 먼저 작성해줘야 하는 것은 해당 메서드가 반환할 데이터의 타입을 작성해주어야 한다.
하지만 모든 메서드가 반드시 값을 되돌려 줘야 하는 것은 아니다. 내부적으로 기능만 수행을 하고 반환값이 없을수도 있는데 이럴땐 어떻게 해야 할까?

```
//기능(메서드)
void TurnOn()
{
    power = true;
}
```

그림 20

TurnOn() 메서드를 보면 반환하는데이터의 타입이 void로 작성되어 있으며 메서드의 내부는 return 키워드가 없다.
void라는 것은 메서드가 반환할 값이 없을 때 작성하며 메서드 내부에서 return을 적지 않아도 된다.
다만 return을 사용 해도 상관은 없으며 반환타입이 있는 메서드와 마찬가지로 return을 작성한다면 그 이후의 코드들은 실행되지 않는다.

```
void TurnOn()
{
    if(state.equals("고장"))
        return;

    power = true;

}
```

void라도 특정라인에서 함수를 종료하고 싶을때 return을 사용할수있다.

마. 객체 배열

지난시간에 같은 타입의 변수 여러개를 동시에 만들고 활용하는 방법으로 배열을 배웠다. 객체역시로 배열로 만들어 관리 할 수 있다.

기본적인 배열 선언 방법과 똑같으며 데이터타입이 만들고자 하는 객체의 클래스이름으로 되어 있다.

1) 객체배열의 초기화

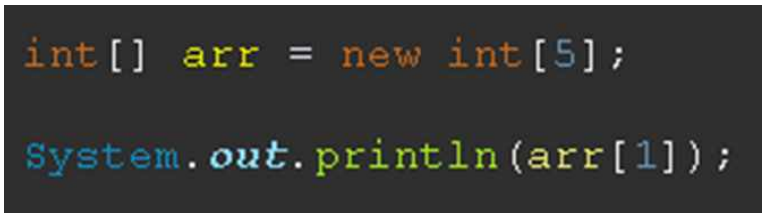
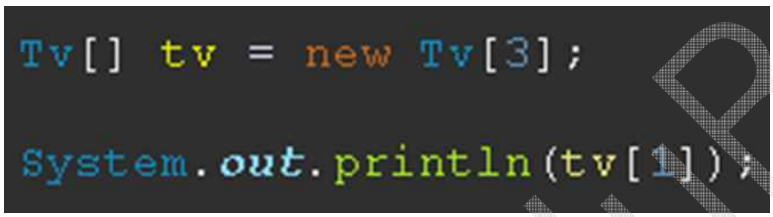


그림23과 같이 기본형의 배열일 경우 요소들을 초기화 하지 않아도 사용 할 수 있으며 데이터타입의 기본값이 들어있다. 예제에서는 요소의 타입이 int 이기에 0 이 출력된다.

그림 23



이와 똑같이 객체배열을 만든 후 요소를 초기화 하지 않고 출력해보면 null이 출력되는 것을 알 수 있다. new Tv[3];은 배열을 생성한 것이지 객체 배열의 요소들에 객체를 넣어둔 것이 아니다.

그림 24

그러므로 그림25와 같이 tv배열의 요소 하나하나에 객체를 생성해서 넣어주어야 한다.

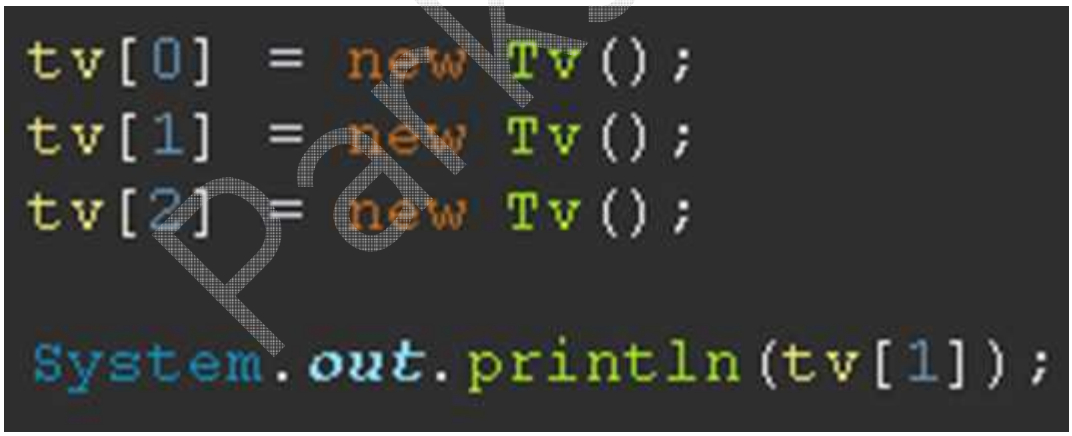


그림 25

2) 객체배열의 사용

```
Tv[] tv = new Tv[3];

tv[0] = new Tv();
tv[1] = new Tv();
tv[2] = new Tv();

tv[0].color = "RED";
tv[0].TurnOn();


System.out.println(tv[0].color);
```

그림 26

객체배열을 사용 하는 방법은 간단하다. 배열의 요소 하나하나에 객체가 들어가 있는것이기에 배열에서 배운 인덱스를 활용하여 요소에 접근한뒤 일반적인 객체를 사용하는 방법과 동일 하다.

다만 만약에 객체배열만 생성후 요소에 객체를 넣어두지 않은채 사용한다면 NullPoint 에러가 발생하며 프로그램이 종료되니 주의하도록 하자.

3. 메서드의 활용

지금까지 기본적인 메서드의 정의와 사용법을 알아보았다. 지금부터는 조금더 깊게 메서드에 대해 다뤄보도록 하자.

가. 재귀호출

재귀호출이란 메서드내부에서 자기자신을 다시 호출하여 반복적으로 무한히 호출되는 것을 말한다.
물론 아무런 조건없이 무한정 반복된다면 무한루프이므로 특정 조건일 때 빠져 나올수 있도록 해야 한다.

```
void method()
{
    System.out.println("메서드 호출");
    //자기자신을 내부에서 다시 호출한다.
    method();
}
```

그림 27

그림27과 같이 메서드 내부에서 자기 자신을 다시 호출하는 구조이다. method()는 println("메서드 호출")을 실행하고 자기 자신을 다시 호출한다. 이때 호출을 하는 메서드는 계속 열려져 있는 채 메서드가 새롭게 실행되는 것이다. 새롭게 호출된 메서드는 또다시 println("메서드 호출")를 실행하고 자기 자신을 다시 호출한다. 이것이 무한히 반복되는 것이다.

그러나 이렇게 종료되는 조건을 따로 작성하지 않고 재귀호출을 한다면 무한루프에 빠지며 메모리를 순식간에 잡아먹으며 프로그램이 멈추게 될 것이다.

따라서 재귀호출은 반드시 빠져 나갈수 있는 조건을 주어야만 한다.

```
void method(int a)
{
    if(a ==0)
        return ;


    System.out.println("메서드 호출"+a);

    //자기자신을 내부에서 다시 호출한다.
    method(--a);
}
```

그림 28 a가0일 때 메서드는 종료된다.

나. 반복문 VS 재귀호출

반복문과 재귀호출 둘다 특정 코드를 반복시킬수 있는 기능을 한다. 그렇다면 두가지의 차이점은 무엇일까?
재귀호출은 메서드가 열려져 있는채 다시 자기 자신을 호출하는 구조 이므로 반복이 될 때마다 열려져 있는 메서드가 계속 늘어나는구조이다. 따라서 성능과 메모리를 많이 소모하는 구조이다.

반면에 반복문은 단순히 특정 영역의 코드를 반복해서 실행하는것이므로 실행이 많아질수록 성능이 느려지는 것은 아니다.

다음은 재귀호출을 활용하여 팩토리얼을 계산하는 예제이다.
팩토리얼이란 !5 과 같이 표기하며 이것의 의미는 1*2*3*4*5를 수행 하라는것과 같은 의미이다.

```
class Test
{

    int factorial(int n)
    {
        if(n ==1)
            return 1;


        return n*factorial(n-1);

    }
}

public class Main {

    public static void main(String[] args) {

        Test t1 = new Test();
        //!5 를 계산한다.
        int result = t1.factorial(5);

        System.out.println(result);
    }

}
```

그림 29

최초로 factorial이 호출될 때 매개변수로 5가 넘어오며 n 매개변수에 5가 저장이된다.
if문은 false 이므로 return을 실행하는데 return 해야 하는값이 n*factorial(n-1)로 자기자신을 다시 호출하는 것이다.

return은 factorial(n-1)의 수행이 다 되어야지만 수행 될 수 있으므로 factorial(4)를 실행한다.

두 번째 호출되는 factorial내부에서 n의 값은 4가 되며 if문은 또다시 false가 된다. 그리고 return을 수행할 때 factorial(4-1)을 실행하게 되며 factorial(4-1)가 실행이 끝나야지 return을 수행할수 있다.

그림30은 이렇게 재귀호출이 수행되는 과정을 매개변수를 적어가며 보여주고 있다.

그림 30

재귀호출은 구조상 쉽게 이해하기 힘든 방식의 메서드 활용법이다. 그리고 성능 또한 좋지 못하다. 그렇다면 재귀호출은 쓰지 말아야 하는것일까? 그렇지는 않다.

방금전의 팩토리얼 예제를 for문을 이용해 구현 할 수 있겠는가? 아마 재귀호출방식보다 코드가 복잡해질 것이다. 이렇게 특정한 경우에 반복문보다 코드가 간결하게 나 올수 있다는 장점이 있다. 트리구조나 메뉴 같은것들을 만들 때 재귀호출을 이용한다면 상당히 편리하게 개발을 할 수 있다.

다. 메서드 오버로딩

메서드 오버로딩은 초반에 잠깐 언급을 한적이 있다. 메서드 이름을 동일하나 그 외의 시그니처(매개변수의 개수,타입)을 다르게 주어 메서드를 여러개 정의하는 것이다.

```
int add(int x, int y)
{
    return x+y;
}

int add(int x,int y,int z)
{
    return x+y+z;
}
```

그림 31

그림31과 같이 add메서드가 이름을 동일하지만 매개변수 부분이 서로 달라 같은 클래스 내에 둘 다 정의가 가능하다. 이렇게 오버로딩을 함으로써 장점은 비슷한 기능들에 대해 동일한 메서드이름을 사용 할 수 있으므로 개발이 편리하며 메서드를 사용할 때 역시 메서드 이름을 일관성 있게 사용할수 있으며 마치 add메서드 한 개가 여러 가지의 기능을 지원해주는 것처럼 사용 할 수 있다.

라. 가변인자

앞서 메서드 오버로딩을 이용해 add메서드의 매개변수를 여러개 받을수 있도록 만든다고 했을 때 불편한 점이 한가지 있다.
바로 매개변수의 개수가 늘어 날 때마다 오버로딩을 통해 계속 메서드를 늘려야 한다는 것이다.만약 20개의 숫자를 더하는 add 메서드를 만드려면 매개변수를 20개를 만들어야 한다.

이럴 때 사용 할 수 있는 것이 가변인자이다.

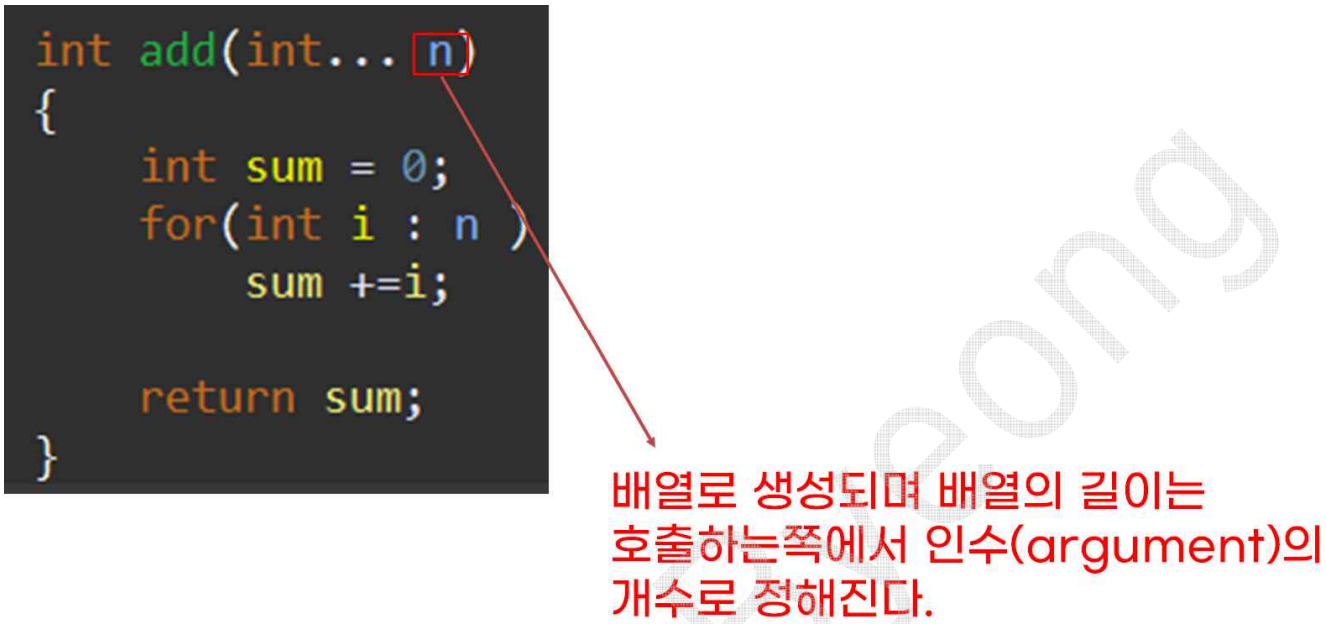


그림 32

매개변수의 타입옆에 ...을 작성하며 변수의 이름을 작성하면된다.
그림32의 n변수는 가변인자이며 배열로 만들어진다.

따라서 향상된 for문을 이용해 요소들을 가져와 합계를 더 할 수 있다.

그림33은 이렇게 가변인자를 이용하여 만든 메서드를 사용하는 방법이다. 매개변수의 개수가 제한이 없다.

```
Test t1 = new Test();
t1.add(10,32);
t1.add(10,32,50);
t1.add(10,32,3,2);
t1.add(10,32,32,1,2,3);
t1.add(10,32,32,1,2,3,3,2,1,3,54,65,3,7,5,4);
```

그림 33