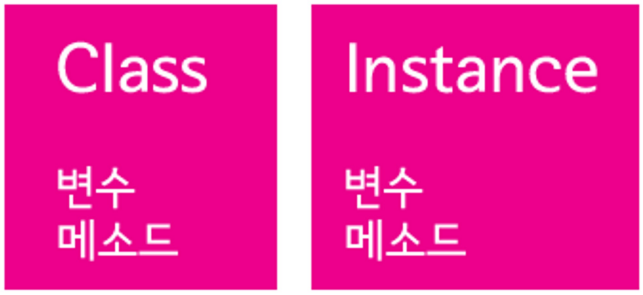
Class Member And Instance Member

Member

맴버(member)는 영어로 구성원이라는 뜻이다. 객체도 구성원을 가지고 있는데, 아래와 같다.

- Variable
- Method



Instance Member - Non-Static Variable, Method

```
class Calculator {
	// instance의 소유인 Variable, instance마다 값이 다르다.
	int left, right;

	public void setOprands(int left, int right) {
		this.left = left;
		this.right = right;
	}

	// Instance Method
	public void sum() {
		System.out.println(this.left + this.right);
	}

	// Instance Method
	public void avg() {
		System.out.println((this.left + this.right) / 2);
	}
}

public class CalculatorDemo1 {
	public static void main(String[] args) {
		Calculator c1 = new Calculator();
		c1.setOprands(10, 20);
		c1.sum();
		c1.avg();

		Calculator c2 = new Calculator();
		c2.setOprands(20, 40);
		c2.sum();
		c2.avg();
	}
}
```

Class Member - Static Variable(field)

```
// 원주율 추가 계산기
class Calculator {

	// Class Member - Variable
	static double PI = 3.14;
```

```
	/*
		인스턴스마다 다른 값을 가지고 있을 필요가 없다.
		Static : Class의 소속
		해당 클래스로부터 생성된 모든 인스턴스가 공유하는 변수
	*/

	int left, right;

	public void setOprands(int left, int right) {

		this.left = left;
		this.right = right;
	}

	public void sum() {
		System.out.println(this.left + this.right);
	}

	public void avg() {
		System.out.println((this.left + this.right) / 2);
	}
}

public class CalculatorDemo1 {
	public static void main(String[] args) {
		// Instance를 통한 접근
		Calculator c1 = new Calculator();
		System.out.println(c1.PI);

		Calculator c2 = new Calculator();
		System.out.println(c2.PI);

		// Class를 통한 직접 접근
		System.out.println(Calculator.PI);
	}
}
-----------------------------------------------------------------------------------------------------
class Calculator2 {
	static final double PI = 3.14; // final : 변수의 상수화!
	static int base = 0;
	int left, right;

	public void setOprands(int left, int right) {
		this.left = left;
		this.right = right;
	}

	public void sum() {
		System.out.println(this.left + this.right+base);
	}

	public void avg() {
		System.out.println((this.left + this.right+base) / 2);
	}
}

public class CalculatorDemo2 {
	public static void main(String[] args) {
		Calculator2 c1 = new Calculator2();
		c1.setOprands(10, 20);
		c1.sum(); // 30

		Calculator2 c2 = new Calculator2();
```

```
        c2.setOprands(20, 40);
        c2.sum(); // 60

        Calculator2.base = 10;

        // class Member인 bass는 c1, c2에 모두 영향을 미친다.
        c1.sum(); // 40
        c2.sum(); // 70
    }
}
```

클래스 변수의 용도

- 인스턴스에 따라서 변하지 않는 값이 필요한 경우 (위의 예에서는 PI)
- 인스턴스를 생성할 필요가 없는 값을 클래스에 저장하고 싶은 경우
- 값의 변경 사항을 모든 인스턴스가 공유해야 하는 경우

Class Member - Static Method

```
class Calculator3 {

// Class Method
    public static void sum(int left, int right) {
        System.out.println(left + right);
    }

    // Class Method
    public static void avg(int left, int right) {
        System.out.println((left + right) / 2);
    }

}

public class CalculatorDemo3 {
    public static void main(String[] args) {
        /*
            인스턴스가 등장하지 않는다.
            Class에 직접 접근하여 Method를 호출하고 있다.
        */
        Calculator3.sum(10, 20);
        Calculator3.avg(10, 20);

        Calculator3.sum(20, 40);
        Calculator3.avg(20, 40);
    }
}
```

Class Member와 Instance Member간의 관계

- **인스턴스는 클래스 멤버에 접근할 수 있다.**
- **인스턴스 메소드는 클래스 멤버에 접근할 수 있다.**

- **클래스는 인스턴스 멤버에 접근할 수 없다.**
- **클래스 메소드는 인스턴스 멤버에 접근할 수 없다.**

클래스는 언제나 인스턴스보다 먼저 존재한다. 클래스를 통해서 인스턴스를 생성하기 때문이다.
클래스(메소드)가 인스턴스 멤버에 접근하는 것은 **아직 생성되지 않은 인스턴스에 접근하려는 것과 마찬가지다.**
하지만 **인스턴스는 반드시 클래스가 존재해야 생성되기 때문에 클래스 멤버에 접근 가능하다.**

```
class C1{
```

```
	static int static_variable = 1;
	int instance_variable = 2;

	static void static_static(){
		System.out.println(static_variable);
	}

	static void static_instance(){
		// 클래스 메소드에서는 인스턴스 변수에 접근 할 수 없다.
		//System.out.println(instance_variable);
	}

	void instance_static(){
		// 인스턴스 메소드에서는 클래스 변수에 접근 할 수 있다.
		System.out.println(static_variable);
	}

	void instance_instance(){
		System.out.println(instance_variable);
	}
}

public class ClassMemberDemo {
	public static void main(String[] args) {
		C1 c = new C1();

		/************************ 인스턴스를 이용한 접근 ************************/
		// 인스턴스를 이용해서 정적 메소드에 접근 -> 성공
		// 인스턴스 메소드가 정적 변수에 접근 -> 성공
		c.static_static();

		// 인스턴스를 이용해서 정적 메소드에 접근 -> 성공
		// 정적 메소드가 인스턴스 변수에 접근 -> 실패
		c.static_instance();

		// 인스턴스를 이용해서 인스턴스 메소드에 접근 -> 성공
		// 인스턴스 메소드가 클래스 변수에 접근 -> 성공
		c.instance_static();

		// 인스턴스를 이용해서 인스턴스 메소드에 접근 -> 성공
		// 인스턴스 메소드가 인스턴스 변수에 접근 -> 성공
		c.instance_instance();
		/*****************************************************************************/

		/************************ 클래스에 직접 접근 ************************/

		// 클래스를 이용해서 클래스 메소드에 접근 -> 성공
		// 클래스 메소드가 클래스 변수에 접근 -> 성공
		C1.static_static();

		// 클래스를 이용해서 클래스 메소드에 접근 -> 성공
```

```
        // 클래스 메소드가 인스턴스 변수에 접근 -> 실패
        C1.static_instance();

        // 클래스를 이용해서 인스턴스 메소드에 접근 -> 실패
        C1.instance_static();

        // 클래스를 이용해서 인스턴스 메소드에 접근 -> 실패
        C1.instance_instance();
    }
    /**************************************************************************/
}
-----------------------------------------------------------------------------------------------------------
```