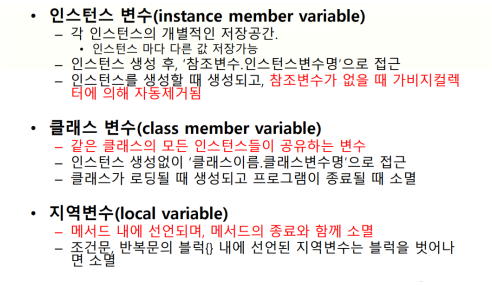
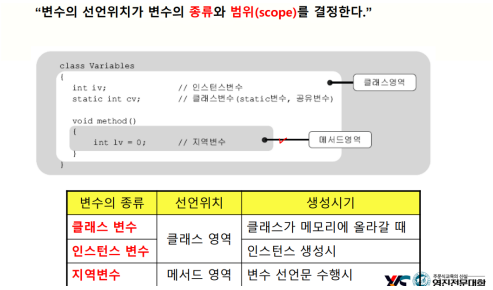
12.28 水 _ Method





🆕생성 시점

Instance 멤버	new 연산자를 이용해서 객체를 생성했을 때 메모리에 올라간다 / **Heap 영역**
class 멤버	처음 사용될 때 메모리에 올라간다 (사용하지 않는 메모리를 지정할 필요 x) / **Code Area**

☠️소멸

Instance 멤버	객체가 소멸될 때 까지 (1. 프로그램 종료 2. 가비지 컬렉터)
class 멤버	처음 사용되고 프로그램 종료 시 까지

Instance or class 멤버 에 접근하는 2가지 방법

1. 객체를 생성하고 참조 변수를 이용해서 멤버에 접근 (`Instance 멤버` , `class 멤버` 둘 다 사용)
2. 객체를 생성 안 하고 class 이름으로 접근 (`class 멤버` 만 사용 가능)

❓ `class` `멤버 변수` 사용 할 때?

1. 객체들 사이에 값이 공유되어야 할 때 class 멤버를 사용
2. 특정 값 자체를 상수 값으로 유지할 때

❓ `class` `멤버 메소드` 사용 왜 하나?

1. 자주 사용하는 메소드를 정의해서 사용하기 위해
   ⇒ 굳이 객체를(알고리즘) 찍을 필요 없다 → 알고리즘 처럼 사용
2. 객체 생성 없이 사용 가능 (메모리 효율성 增)

경우의 수

> class는 계속 사용 가능 (프로그램 종료 될 때 까지 지속되기 때문)

class	→	class	可能
class	→	Instance	**不可** ⇒ Instance 는 만들어 졌는지 여부를 확인 할 수가 없다.
Instance	→	class	可能
Instance	→	Instance	可能

class -> Instance 은 태어나는 시점이 틀리다

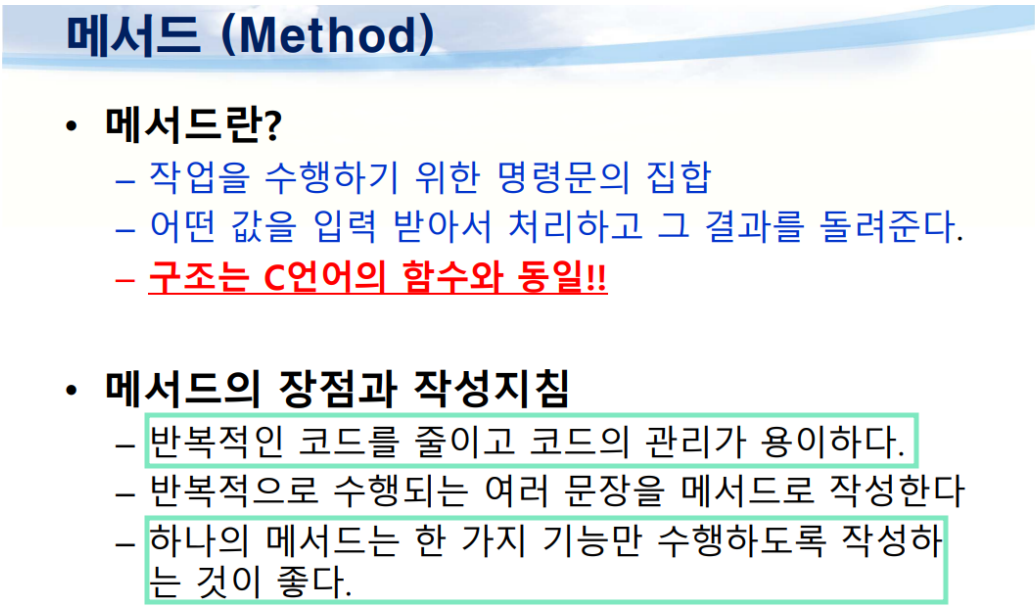
class → Instance

```
main이 포함된 프로젝트는 실행시킬 용도로만 사용하기 !

package test;

class Bar {
```

```
}

// test2.main();
public class test2 {

  int x =3 ;

  // test2 클래스는 실행(생성) 시키지 않는다 !!
  public static void main(String[] args) {
    System.out.println(x); //  => class -> Instance  [ 멤버 변수 ]
   prt1(); // class -> Instance   [멤버 메서드]

  }
  void prt1() {}

}
```

Method

- 프로그래밍을 하는데 단위가 필요하다 ⇒ class
- class 는 변수와 함수의 구성! → 구조적 언어의 함수 개념을 class 에 가져온 것이다
- 기존의 함수 개념을 class 에 집어넣고 멤버로 묶었다.



클래스메서드(static메서드)와 인스턴스메서드

- **인스턴스 메서드**
  - 인스턴스 생성 후, '참조변수.메서드이름()'으로 호출
  - 인스턴스 변수나 인스턴스 메서드와 관련된 작업을 하는 메서드
  - 메서드 내에서 인스턴스 변수 사용가능

- **클래스 메서드(static메서드)**
  - 객체생성없이 '클래스이름.메서드이름()'으로 호출
  - 인스턴스 변수나 인스턴스 메서드와 관련 없는 작업을 하는 메서드
  - 메서드 내에서 인스턴스 변수 사용불가
  - 메서드 내에서 인스턴스 변수를 사용하지 않는다면 static을 붙이는 것을 고려한다.

Method 기본 형태

```
class Bar {
  // 반환형 메서드 이름 (매개변수) {
    // }
```

반환 형으로 올 수 있는 자료형

- primitive (byte ~long • double)
- ⭐⭐⭐참조 변수 ⭐⭐⭐
- void

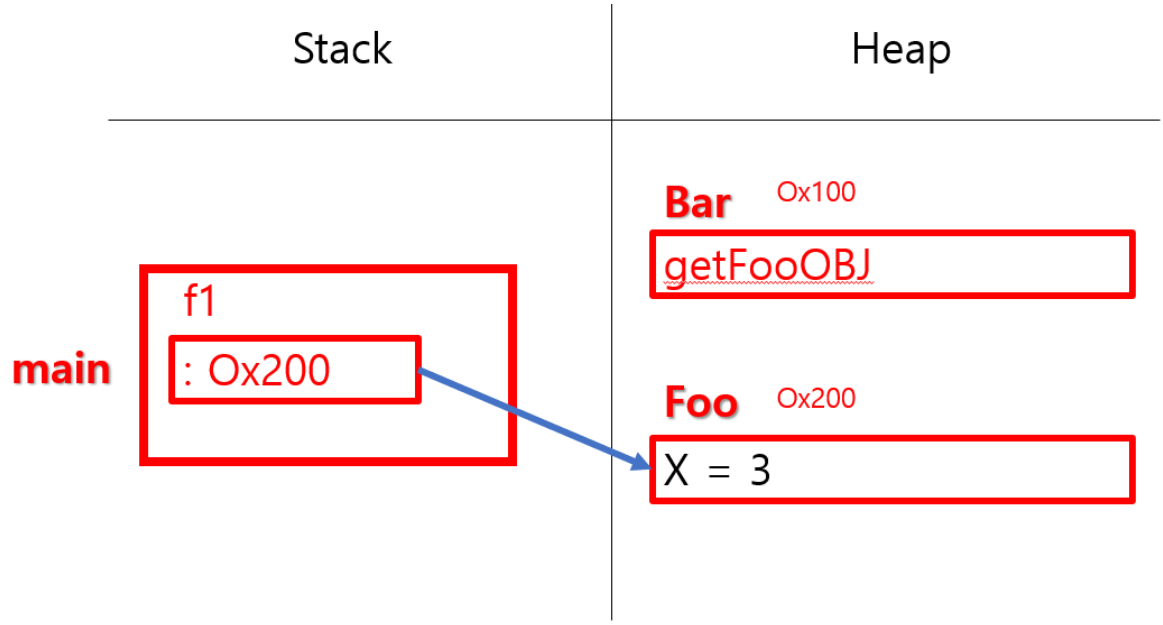
❓ return 값이 없으면 for 와 while 처럼 활용할 수 있다.

```
package test;

class Bar {
```

```
  int x = 3;
  void prt() {System.err.println("hello");}

  static int y = 4;
  static void prt2() {System.out.println("hello");}
}

// test2.main();
public class test2 {

  int x =3 ;

  public static void main(String[] args) {


  }

}
```

반환 형 ⇒ 참조 변수 (주소 값)

```
package test;


class Bar {
  // 반환형 메서드 이름 (매개변수) {
    // }

  Foo getFooOBJ() {
    return new Foo();
  }
}


class Foo {
  int x = 3 ;
}


public class test2 {

  int x =3 ;

  // test2 클래스는 실행 시키지 않는다 !!
  public static void main(String[] args) {
    Foo f1 = (new Bar()).getFooOBJ();   //=> new Bar() 클래스로 찍힌 객체의 주소가 넘어옴

  }
}
```



반환 형 : `원시 데이터 자료형` , `참조 변수 ( class 이름)` , **`void`**

반환 형과 return 해주는 형이 반드시 일치해야 한다!!

배열을 자료형으로 사용한 경우

```
package test;


class Bar {
  // 반환형 메소드 이름 (매개변수) {
    // }

  int[] doSomething() {
    int []myArray = {2,4};
    return myArray;
  }
}

public class test2 {

  int x =3 ;
```

```
  // test2 클래스는 실행 시키지 않는다 !!
  public static void main(String[] args) {


  }
}
```

2 개 이상 반환 하고 싶을 때 - 배열 만들기

```
package test;


class Bar {
  // 반환형 메소드 이름 (매개변수) {
    // }

  static int[] doSomething() {

    int []myArray = {2,4};

    return myArray;
  }
}

public class test2 {

  // test2 클래스는 실행 시키지 않는다 !!
  public static void main(String[] args) {


  }
}
```

2 개 이상 반환 하고 싶을 때 - 객체 만들기

```
package test;


class Bar {
  // 반환형 메소드 이름 (매개변수) {
    // }

  static StdInfo doSomething() {


    return new StdInfo(2,"hello");
//  return null                    =>       null 값이 없다는 표현을 이렇게 나타낸다
```

```
  }
}

class StdInfo {

  int id ;
  String name;

  StdInfo(int argId , String argName ){
    id = argId ;
    name = argName;
  }

}


public class test2 {

  // test2 클래스는 실행 시키지 않는다 !!
  public static void main(String[] args) {


  }
}
```

반환 형이 void

```
package test;


class Bar {
  // 반환형 메소드 이름 (매개변수) {
    // }

  static void doSomething() {

    if ( 3 < 2 )
        return;      // => 반환형이 void일 때 이렇게 아무것도 작동 안하게 할 때 활용 가능

    return ;
  }
}

class StdInfo {

  int id ;
  String name;

  StdInfo(int argId , String argName ){
    id = argId ;
    name = argName;
  }
```

```
}


public class test2 {

  // test2 클래스는 실행 시키지 않는다 !!
  public static void main(String[] args) {


  }
}
```

에러 발생 시

```
package test;

ErrorCode {
  static final int UNNOWN_ERROR = 1;
}


class Bar {
  // 반환형 메소드 이름 (매개변수) {
    // }

  static void doSomething() {

    if ( 3 < 2 )
        return ErrorCode.UNNOWN_ERROR;

    return ;
  }
}

class StdInfo {

  int id ;
  String name;

  StdInfo(int argId , String argName ){
    id = argId ;
    name = argName;
  }

}


public class test2 {

  // test2 클래스는 실행 시키지 않는다 !!
  public static void main(String[] args) {
```

```
  }
}
```

예습

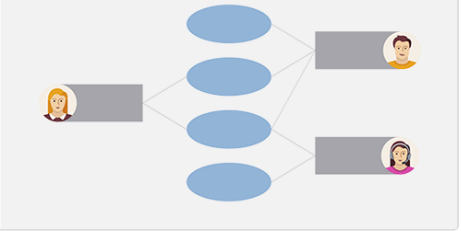
Call - by Value ， Call - by Reference

Initialization block → instance , class

[UML] UML 기초

📌 Unified Modeling Language 프로그램 설계를 표현하기 위해 사용하는 표기법 요구분석, 시스템 설계, 시스템 구현 등의 시스템 개발 과정에서 개발자간의 의사소통을 원활하게 이루어지게 하기 위하여

https://velog.io/@hanblueblue/UML-UML-%EA%B8%B0%EC%B4%88



UML: 클래스 다이어그램과 소스코드 매핑

불과 몇 년 되지 않은 학생 시절... 처음으로 UML을 접했고, UML의 기초적인 그리는 법과 사용법을 배웠습니다. 개인적으로 쉽지 않은 수업이었는데 그 중 가장 많이 사용되는 클래스 다이어그램에서 클래스

https://www.nextree.co.kr/p6753/