



It's Your Life

with





자바의 메모리

스택
영역

메서드
영역

힙
영역



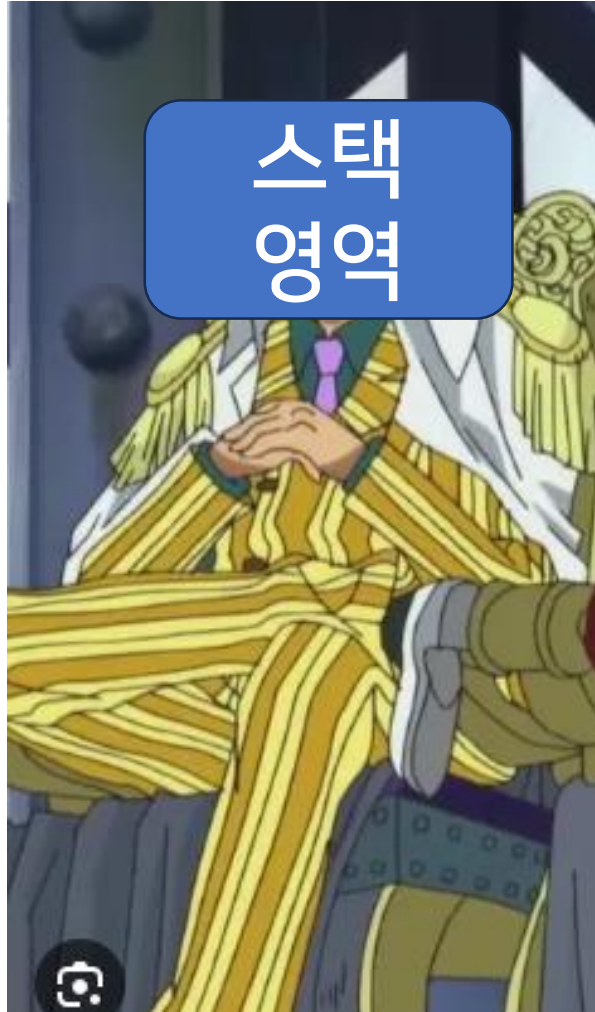




메서드
영역



스택
영역



힙
영역





메서드
영역



자바 프로그램의 공통 데이터 관리

클래스에 관한 정보를 보관

static 변수 보관

상수 관련 데이터 보관



스택
영역

프로그램이 실행되면 하나 생성

메서드가 호출 되면 하나씩 쌓인다

메서드가 종료되면 사라진다



클래스에 의해 생성된 인스턴스 보관

배열도 생성되어 보관

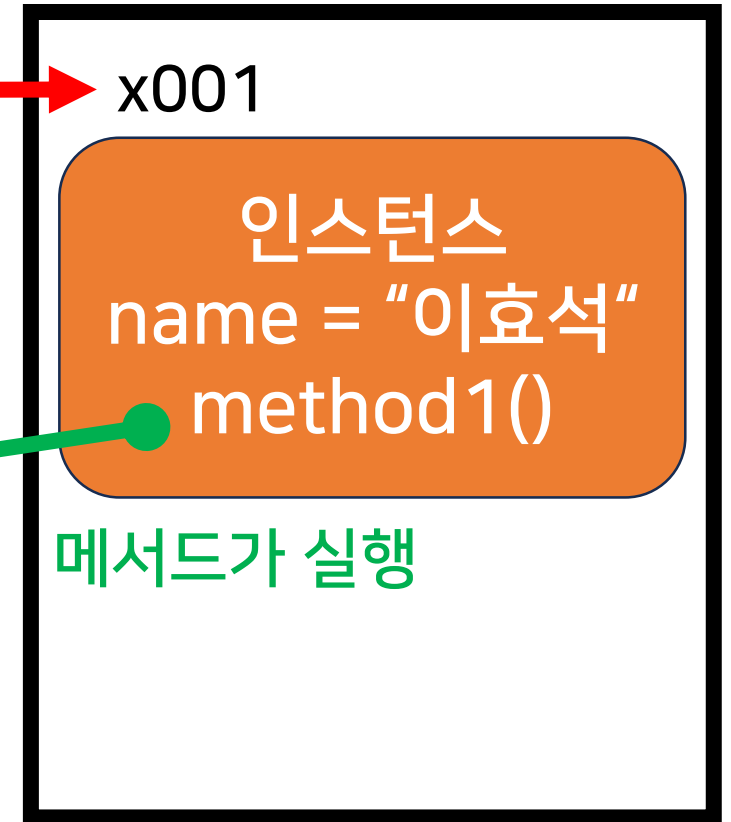
Garbage Collector에 의해
청소가 이루어지는 공간



메서드 영역

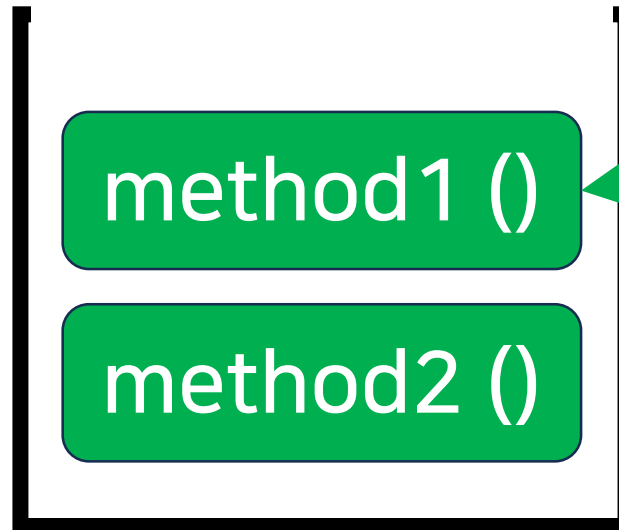


힙 영역



인스턴스화

스택 영역





왜 메서드 영역이죠?

method 를 모든 인스턴스에
따로 만들 필요가 있을까요?

메서드 영역

```
class Parent {  
  int value = 1;  
  
  void method()  
}
```

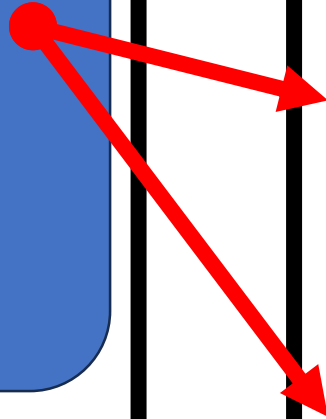
힙 영역

x001

인스턴스

x002

인스턴스





메서드 영역

```
class Parent {  
    void method()  
}
```

힙 영역

x001

value = 1;


x002

value = 1

method1 실행 요청

method1 실행 요청



사진 제공자 : 



메서드 영역

클래스 정보

Static 영역

상수 영역



스택과 힙 영역

```
public class Data { no usages
    public int value;

    Data(int value) { no usage
        this.value = value;
    }

    public int getValue() { n
        return this.value;
    }
}
```




```
public class MemoeyMain1 { new *  
    public static void main(String[] args) { new *  
        System.out.println("메인 메서드 시작");  
        method1();  
        System.out.println("메인 메서드 종료");  
    }  
}
```

main 메서드 부터 수행 시작 후,
method1 호출

```
static void method1() { 1 usage new *  
    System.out.println("메서드1 수행 시작");  
    Data data1 = new Data( value: 10);  
    method2(data1);  
    System.out.println("메서드1 수행 종료");  
}
```

method1 수행 시작 후,
Data 클래스 인스턴스 화 후
method2 호출

```
static void method2(Data data) { 1 usage new *  
    System.out.println("메서드2 수행 시작");  
    System.out.println("data 의 value 값은 : " + data.getValue());  
    System.out.println("메서드2 수행 종료");  
}
```

```
}
```



```
public class MemoeyMain1 { new *
    public static void main(String[] args) { new *
        System.out.println("메인 메서드 시작");
        method1();
        System.out.println("메인 메서드 종료");
    }

    static void method1() { 1 usage new *
        System.out.println("메서드1 수행 시작");
        Data data1 = new Data(value: 10);
        method2(data1);
        System.out.println("메서드1 수행 종료");
    }

    static void method2(Data data) { 1 usage new *
        System.out.println("메서드2 수행 시작");
        System.out.println("data 의 value 값은 : " + data.getValue());
        System.out.println("메서드2 수행 종료");
    }
}
```

method2 수행 후,
Data 인스턴스의 값을
가져와서 출력

어떤 순서로 출력이 나올지 예상이 되시는 분!?



```
public class MemoeyMain1 { new *
    public static void main(String[] args) { new
        System.out.println("메인 메서드 시작");
        method1();
        System.out.println("메인 메서드 종료");
    }

    static void method1() { 1 usage new *
        System.out.println("메서드1 수행 시작");
        Data data1 = new Data(value: 10);
        method2(data1);
        System.out.println("메서드1 수행 종료");
    }

    static void method2(Data data) { 1 usage new
        System.out.println("메서드2 수행 시작");
        System.out.println("data 의 value 값은 : ");
        System.out.println("메서드2 수행 종료");
    }
}
```

스택 영역

A diagram illustrating a stack frame. It consists of a large, empty rectangular box with a thick black border. At the bottom of this box is a smaller, solid green rounded rectangle. Inside the green rectangle, the text "main()" is written in white. A red arrow originates from a red dot on the line "public static void main(String[] args) {" in the code block on the left and points directly to the green "main()" box.

main()


```
public class MemoeyMain1 { new *
    public static void main(String[] args) { new
        System.out.println("메인 메서드 시작");
        method1();
        System.out.println("메인 메서드 종료");
    }

    static void method1() { 1 usage new *
        System.out.println("메서드1 수행 시작");
        Data data1 = new Data(value: 10);
        method2(data1);
        System.out.println("메서드1 수행 종료");
    }

    static void method2(Data data) { 1 usage new
        System.out.println("메서드2 수행 시작");
        System.out.println("data 의 value 값은 : ");
        System.out.println("메서드2 수행 종료");
    }
}
```

스택 영역



method1()

main()

```

public class MemoeyMain1 { new *
    public static void main(String[] args) { new
        System.out.println("메인 메서드 시작");
        method1();
        System.out.println("메인 메서드 종료");
    }

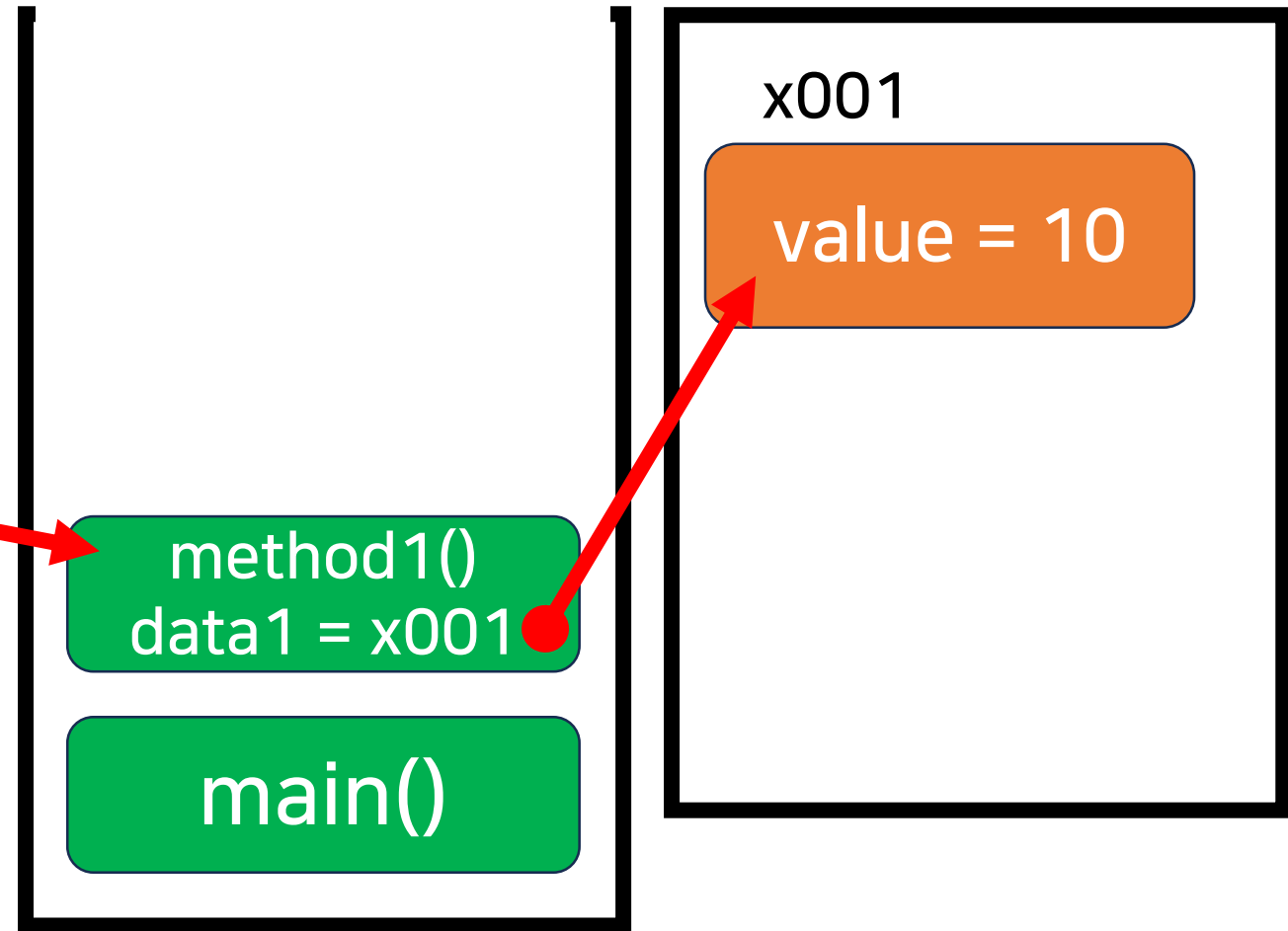
    static void method1() { 1 usage new *
        System.out.println("메서드1 수행 시작");
        Data data1 = new Data(value: 10);
        method2(data1);
        System.out.println("메서드1 수행 종료");
    }

    static void method2(Data data) { 1 usage new
        System.out.println("메서드2 수행 시작");
        System.out.println("data 의 value 값은 : ");
        System.out.println("메서드2 수행 종료");
    }
}

```

스택 영역

힙 영역



```

public class MemoeyMain1 { new *
    public static void main(String[] args) { new
        System.out.println("메인 메서드 시작");
        method1();
        System.out.println("메인 메서드 종료");
    }

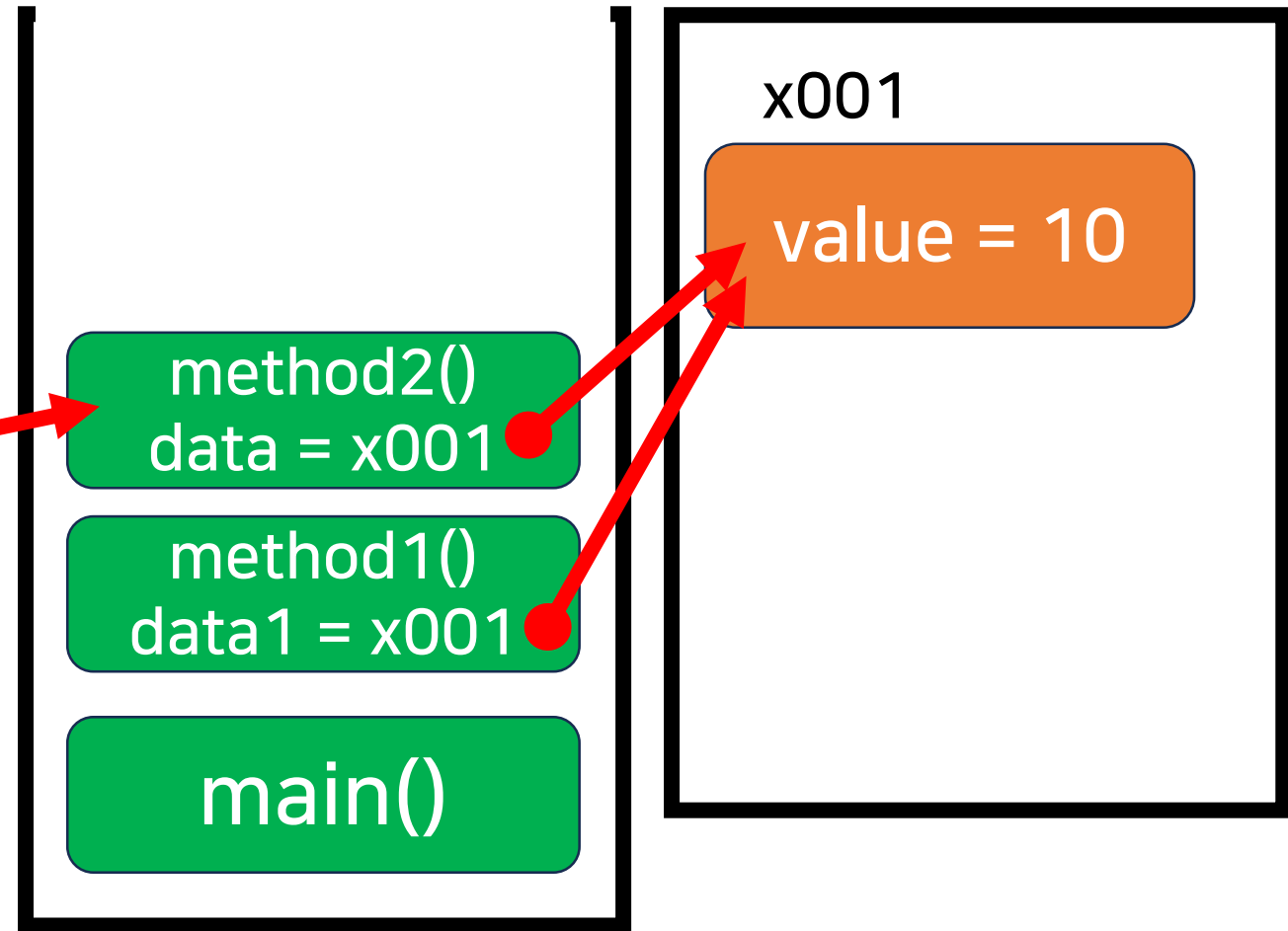
    static void method1() { 1 usage new *
        System.out.println("메서드1 수행 시작");
        Data data1 = new Data(value: 10);
        method2(data1);
        System.out.println("메서드1 수행 종료");
    }

    static void method2(Data data) { 1 usage new
        System.out.println("메서드2 수행 시작");
        System.out.println("data 의 value 값은 : ");
        System.out.println("메서드2 수행 종료");
    }
}

```

스택 영역

힙 영역



```

public class MemoeyMain1 { new *
    public static void main(String[] args) { new
        System.out.println("메인 메서드 시작");
        method1();
        System.out.println("메인 메서드 종료");
    }

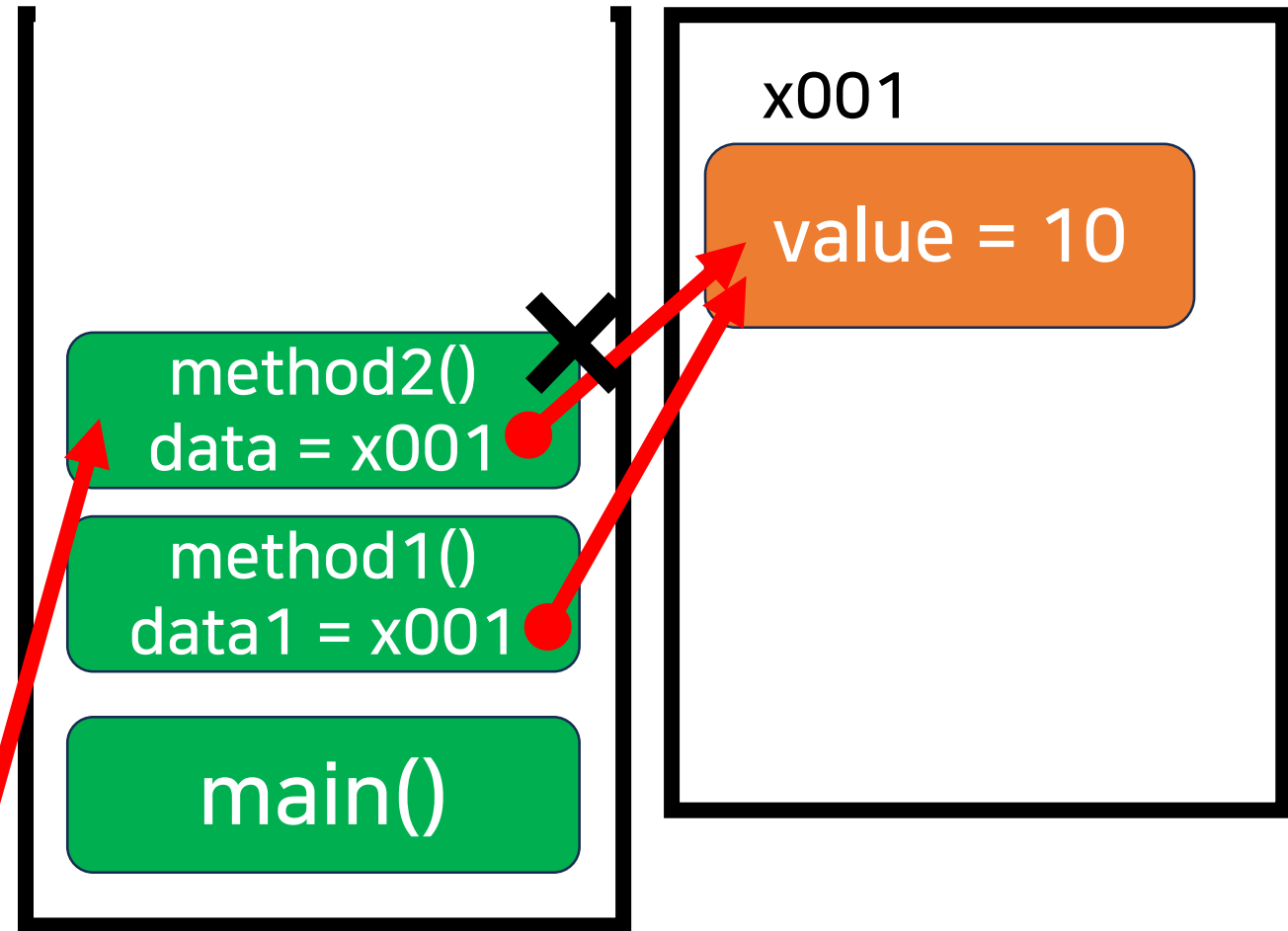
    static void method1() { 1 usage new *
        System.out.println("메서드1 수행 시작");
        Data data1 = new Data(value: 10);
        method2(data1);
        System.out.println("메서드1 수행 종료");
    }

    static void method2(Data data) { 1 usage new
        System.out.println("메서드2 수행 시작");
        System.out.println("data 의 value 값은 : ");
        System.out.println("메서드2 수행 종료");
    }
}

```

스택 영역

힙 영역




```

public class MemoeyMain1 { new *
    public static void main(String[] args) { new
        System.out.println("메인 메서드 시작");
        method1();
        System.out.println("메인 메서드 종료");
    }

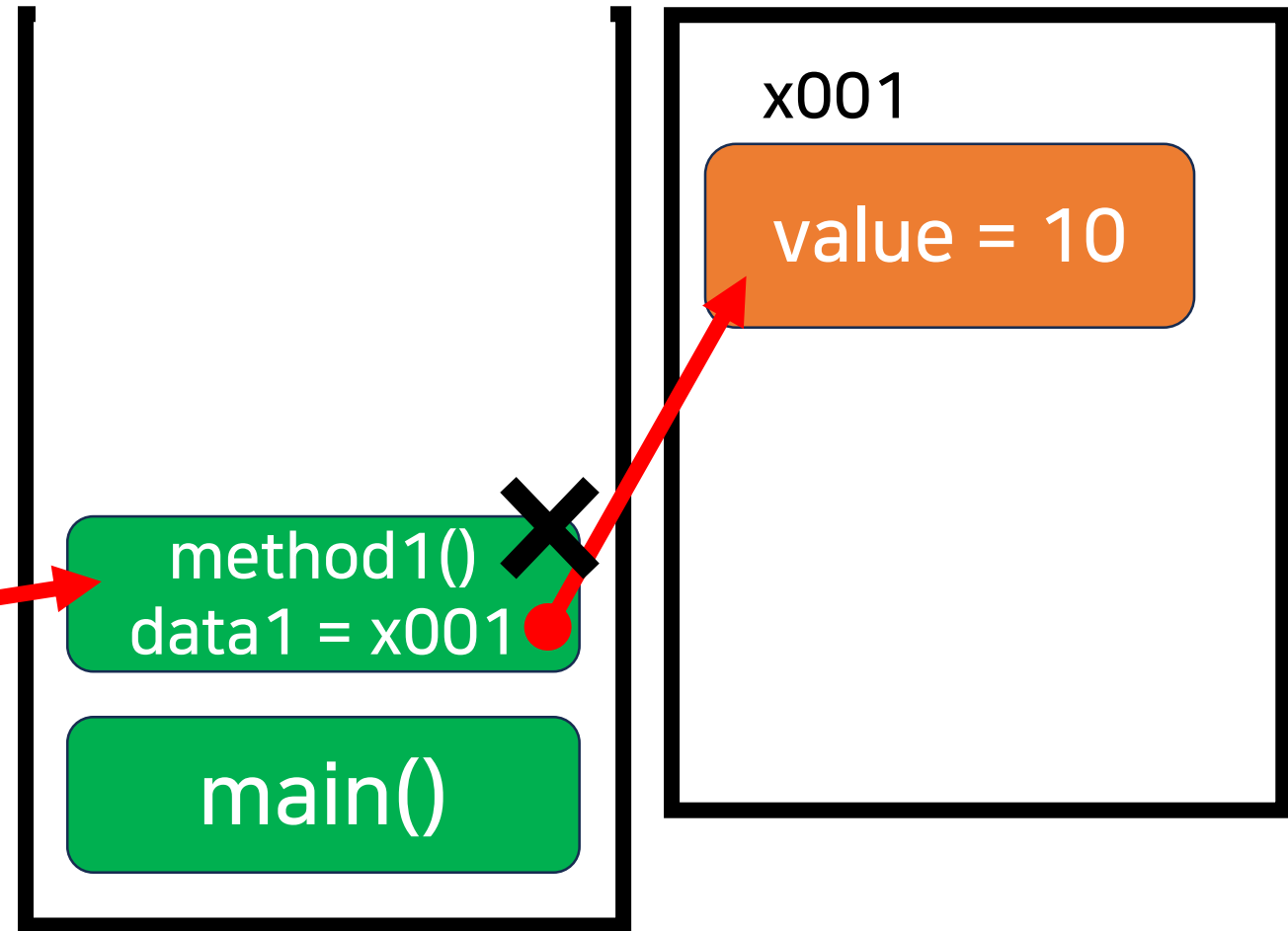
    static void method1() { 1 usage new *
        System.out.println("메서드1 수행 시작");
        Data data1 = new Data(value: 10);
        method2(data1);
        System.out.println("메서드1 수행 종료");
    }

    static void method2(Data data) { 1 usage new
        System.out.println("메서드2 수행 시작");
        System.out.println("data 의 value 값은 : ");
        System.out.println("메서드2 수행 종료");
    }
}

```

스택 영역

힙 영역



```

public class MemoeyMain1 { new *
    public static void main(String[] args) { new
        System.out.println("메인 메서드 시작");
        method1();
        System.out.println("메인 메서드 종료");
    }

    static void method1() { 1 usage new *
        System.out.println("메서드1 수행 시작");
        Data data1 = new Data(value: 10);
        method2(data1);
        System.out.println("메서드1 수행 종료");
    }

    static void method2(Data data) { 1 usage new
        System.out.println("메서드2 수행 시작");
        System.out.println("data 의 value 값은 : ");
        System.out.println("메서드2 수행 종료");
    }
}

```

스택 영역

main()

힙 영역



x001

~~value~~ = 10



```
public class MemoeyMain1 { new *
    public static void main(String[] args) { new
        System.out.println("메인 메서드 시작");
        method1();
        System.out.println("메인 메서드 종료");
    }

    static void method1() { 1 usage new *
        System.out.println("메서드1 수행 시작");
        Data data1 = new Data(value: 10);
        method2(data1);
        System.out.println("메서드1 수행 종료");
    }

    static void method2(Data data) { 1 usage new
        System.out.println("메서드2 수행 시작");
        System.out.println("data 의 value 값은 : ");
        System.out.println("메서드2 수행 종료");
    }
}
```

스택 영역



main()

A diagram illustrating the stack area. A large, empty rectangular box with a black border represents the stack. A red arrow points from the closing curly brace of the `main` method in the code block on the left to a green rounded rectangle labeled `main()` located at the bottom of the stack box.

메인 메서드 시작

메서드1 수행 시작

메서드2 수행 시작

data 의 value 값은 : 10

메서드2 수행 종료

메서드1 수행 종료

메인 메서드 종료





드디어 static!?

개념

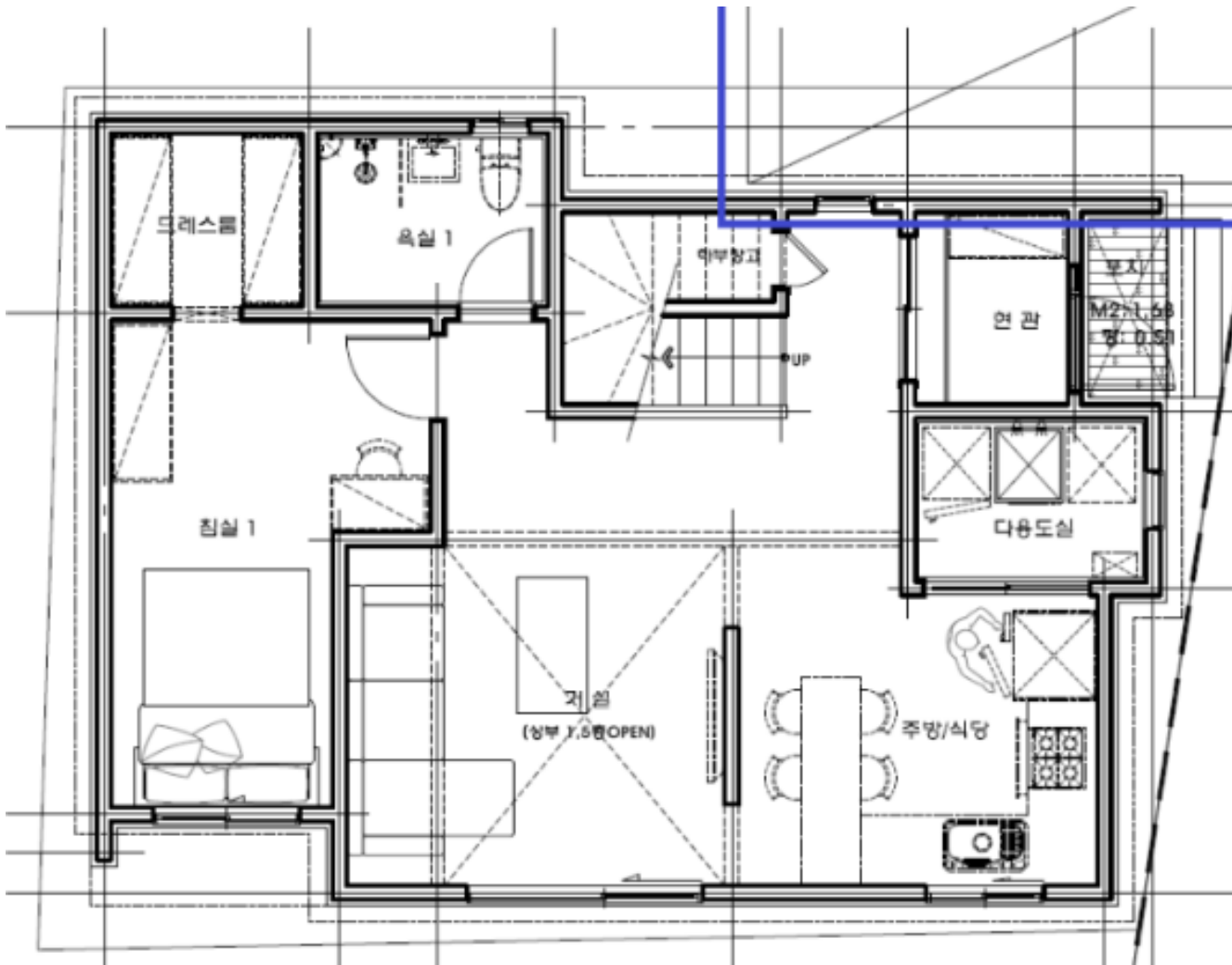


먼저 static이라는 용어를 해석해보면 말그대로 정적 혹은 고정된이란 의미를 가지고 있다.

자바에서는 static 키워드를 사용하여 static 변수와 static 메소드를 생성할 수 있고, 이 둘을 합쳐 정적 멤버라고 한다.



**제가 자바 클래스를
뭐에 자주 비유했죠!?**



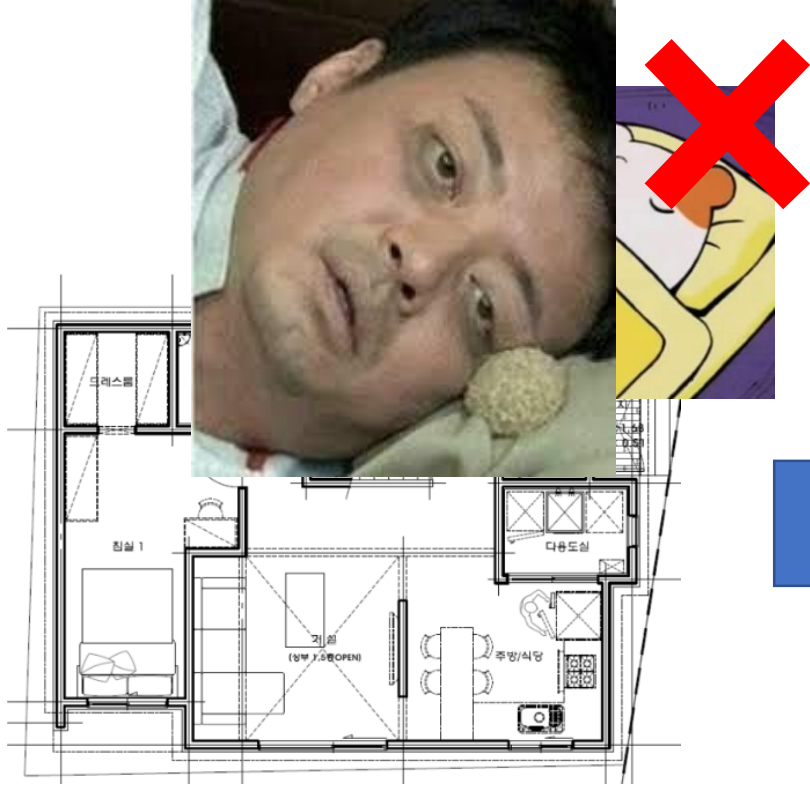
설계 도면은 실체가
존재 하나요?

설계 도면이 있다고 해서
실제 사용 가능한 물건이
제 손에 있는 것이 아닙니다

설계도인 클래스를 실체화



하는 과정이 뭐였죠?



인스턴스화



```
public class Data { 3 usages  
    public int value;
```

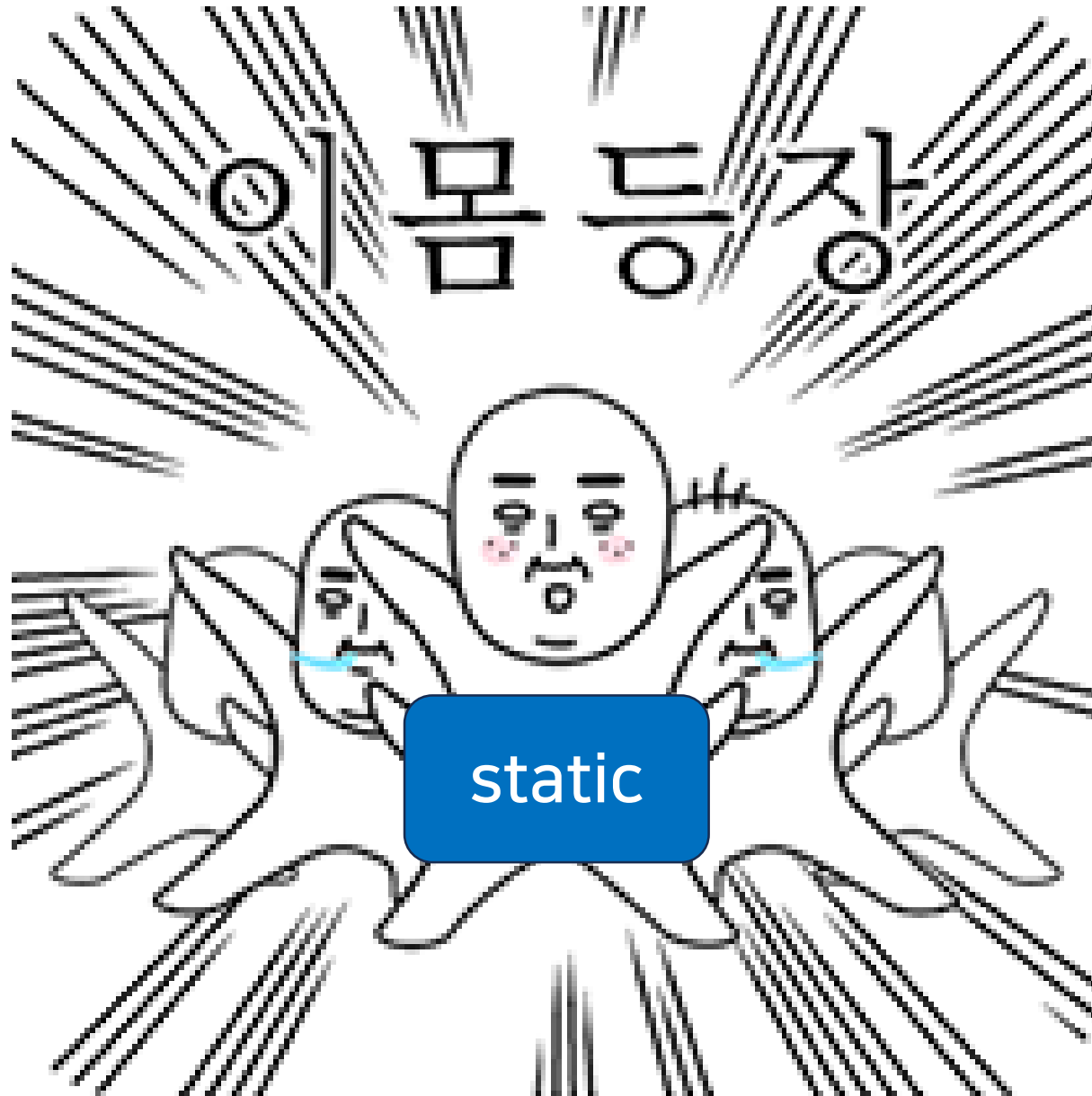
아직 인스턴스화 되기 전이므로
직접 접근 및 사용이 불가능

```
public class MemoeyMain1 { new *  
    public static void main(String[] args) {  
        Data data1 = new Data(value: 10);  
        System.out.println(data1.value);
```

인스턴스화를 통해 실체화를
시킨 이후에나 접근하여 사용 가능

그런데 귀찮게 이런 방식으로만
써야 할까요?

꼭 그렇게





**그럼 변수랑 메서드를
인스턴스가 아니라
클래스 자체에 붙여 버리자!**

static 키워드를 통해서
변수와 메서드를 클래스에
바로 붙이기!

```
public class DataStatic {  
    public static int value = 1;  
    public static void showValue() {  
        System.out.println("으아아아아 난 실체 없이 사용되고 싶어!!!!");  
    }  
}
```



```
public class StaticMain {  
    public static void main(String[] args) {  
        System.out.println(DataStatic.value);  
        DataStatic.showValue();  
    }  
}
```

인스턴스화 없이 바로 사용!!

1

으아아아아 난 실체 없이 사용되고 싶어!!!!!!



```
public class StaticMain { new *  
    public static void main(String[] args) {  
        System.out.println(DataStatic.value);  
        DataStatic.showValue();  
    }  
}
```



운영 클래스에 무지성으로
psvm 을 치던 이유를
지금은 이해가 되실 겁니다!

운영 클래스는 무언가를
만들고 실행해야 하는 부분인데
이 친구는 어디서 인스턴스화를
시키나요!?

그래서 인스턴스화 없이 바로
사용 가능한 static 으로
메서드를 정의

static 이 붙은 변수, 메서드는



이제 메서드 영역이 관리 합니다!

static 을 사용하면 인스턴스화 후 힙 영역에서 사용하는 것이 아니라
바로 static 영역(= 메서드 영역의 내부)에서 사용 가능!



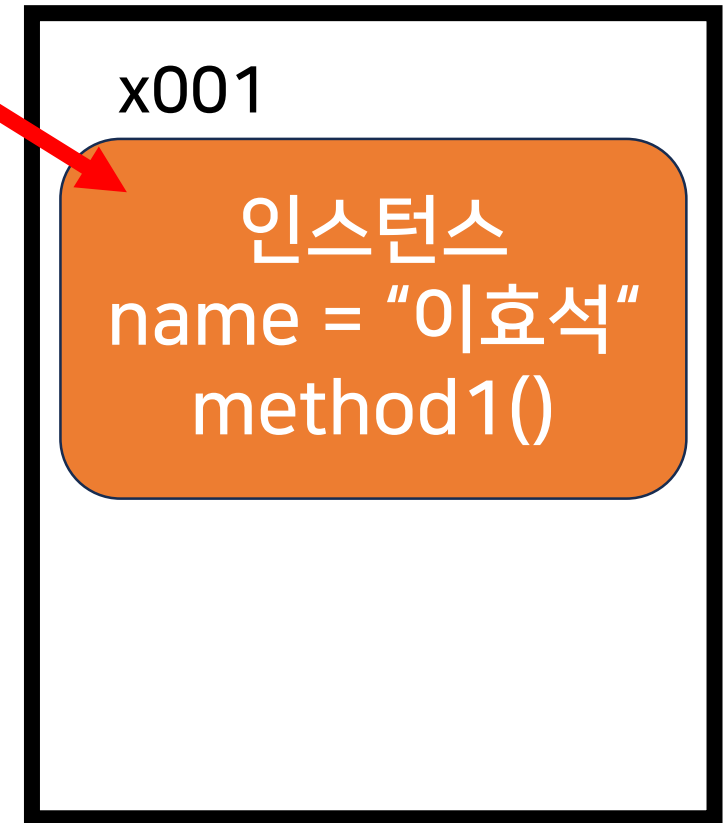
메서드 영역



스택 영역



힙 영역





**보통 static 요소는 클래스 자체와 연관이 있으므로
모든 인스턴스가 요소에 접근 및 공유가 가능하도록
인스턴스화 작업 없이 바로 접근이 가능합니다!**



static 변수

```
public class Counter {  
    public int counter;  
  
    public Counter() {  
        this.counter++;  
    }  
}
```

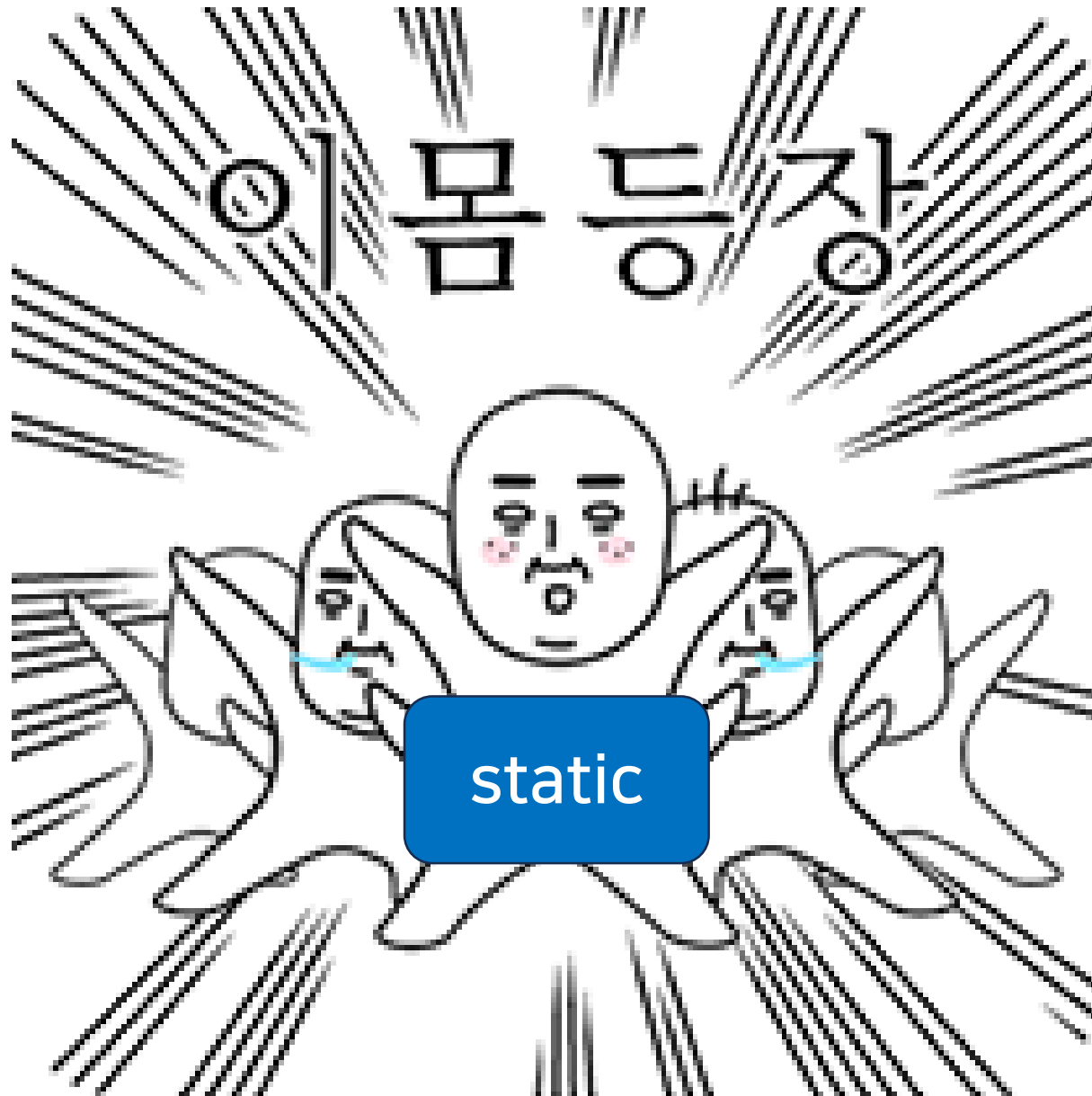
특정 클래스가
인스턴스화 된 횟수를 기록하는
클래스를 만들어 봅시다!

```
public class CounterMain {  
    public static void main(String[] args) {  
        Counter counter1 = new Counter();  
        Counter counter2 = new Counter();  
  
        System.out.println("Counter 가 인스턴스화 된 횟수는 : " + counter2.counter);  
    }  
}
```

어떤 결과가 나올까요!?

Counter 가 인스턴스화 된 횟수는 : 1





```
public class Counter { 4 usages r  
    public static int counter; 2 u  
  
    public Counter() { 2 usages r  
        this.counter++;  
    }  
}
```

counter 멤버를
static 으로 변경!

왜 갑자기 warning 이
났을까요?

```
public class CounterMain {  
    public static void main(String[] args) {  
        Counter counter1 = new Counter();  
        Counter counter2 = new Counter();  
  
        System.out.println("Counter 가 인스턴스화 된 횟수는 : " + counter2.counter);  
    }  
}
```



Counter 가 인스턴스화 된 횟수는 : 2





메서드 영역

힙 영역

```
class Counter {  
    int value;  
  
    Counter() {  
        this.value++;  
    }  
}
```

인스턴스화

인스턴스화

x001

value = 1;

x002

value = 1

우리가 출력한 값

Counter 가 인스턴스화 된 횟수는 : 1

메서드 영역

힙 영역



```
class Counter {  
    static int value = 2;  
  
    Counter() {  
        this.value++;  
    }  
}
```

인스턴스화

인스턴스화

우리가 출력한 값

x001

x002

Counter 가 인스턴스화 된 횟수는 : 2



static 메서드

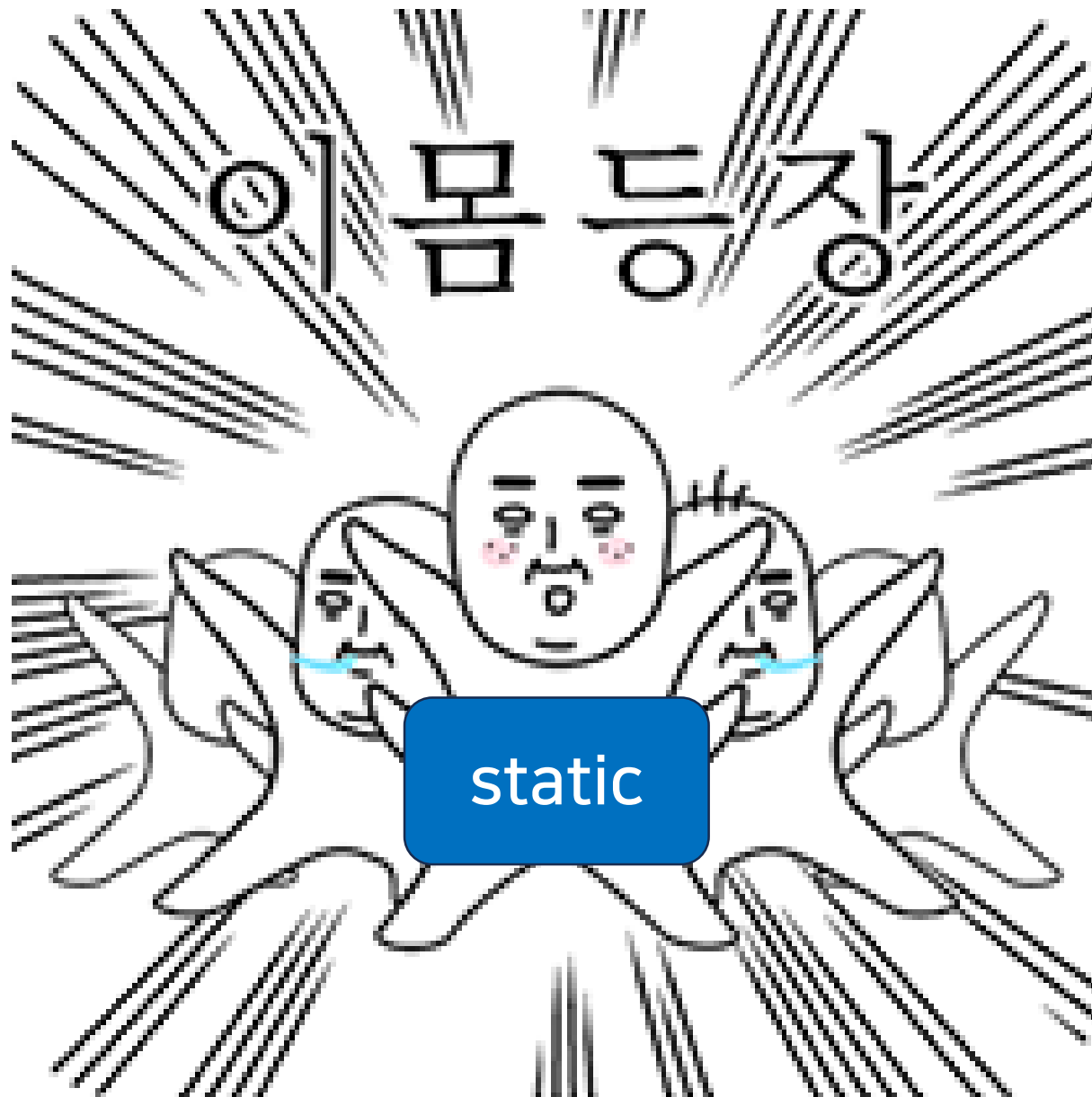
```
public class AddStr { no usages new *  
    public String addStr(String str) {  
        return str + "합니다요!";  
    }  
}
```

특정 문자열을 받아서
해당 문자열 뒤에
'합니다요!' 를 붙여주는 메서드

우리반은 다트 대회 우승을 합니다요!

```
public class AddStrMain {  
    public static void main(String[] args) {  
        String str1 = "우리반은 다트 대회 우승을 ";  
  
        AddStr addStr = new AddStr();  
        System.out.println(addStr.addStr(str1));  
    }  
}
```

공용 기능 메서드지만
이렇게 인스턴스를 만들어서
써야만 합니다 ☹️





```
public class AddStr { 2 usages new *  
    public static String addStr(String str) {  
        return str + "합니다요!";  
    }  
}
```

매서드를 static 으로 변경!

```
public class AddStrMain {  
    public static void main(String[] args) {  
        String str1 = "우리반은 다트 대회 우승을 ";  
        System.out.println(AddStr.addStr(str1));  
    }  
}
```



인스턴스화 작업 없이
바로 클래스에서 메서드를 꺼내서 사용

우리반은 다트 대회 우승을 합니다요!





final 변수



final 변수는 최초 생성 시



단 한번만 값을 설정 할 수 있으며

그 이후로 값이 변하지 않는 변수

```
public class Final { 2 usages    new *  
    final int radius; 2 usages  
    static double PI = 3.14; 1 usage  
  
    public Final(int radius) { 1 usage    new *  
        this.radius = radius;  
    }  
  
    public double getCircleArea() { 1 usage    new *  
        return PI * this.radius * this.radius;  
    }  
}
```

final 타입으로 반지름 멤버를 설정

final 타입이기 때문에
최초 생성시 한번만 값을
설정할 수 있기 때문에
반드시 생성자로 값을 입력

```
public class Final { 2 usages    new *
    final int radius; 3 usages
    static double PI = 3.14; 1 usage

    public Final(int radius) { 1 usage    new *
        this.radius = radius;
    }

    public double getCircleArea() { 1 usage    new *
        return PI * this.radius * this.radius;
    }
}
```

원의 넓이를 구하는 메서드



```
public class FinalMain { new *  
    public static void main(String[] args) { new *  
        Final aFinal = new Final(radius: 3);  
  
        System.out.println(aFinal.getCircleArea());  
    }  
}
```

28.259999999999998



```
public class FinalMain {  
    public static void main(String[] args) {  
        Final aFinal = new Final(radius: 3);  
  
        System.out.println(aFinal.getCircleArea());  
        aFinal.radius = 10;  
    }  
}
```



Cannot assign a value to final variable 'radius'

Make 'Final.radius' not final Alt+Shift+Enter More

반지름의 값을 변경하려고 하면
final 키워드에 의해 에러 발생!



final 상수

final 은 주로



**모든 인스턴스에서 공유할 목적으로
상수 값을 설정할 때 많이 사용합니다!**

```
public class Final2 {  
    int radius;  
    static final double PI = 3.14;  
  
    public double getCircleArea() {  
        return PI * this.radius * this.radius;  
    }  
}
```

상수로 사용할 목적이므로
선언 시에 바로 값을 지정

지정하지 않을 경우
컴파일 에러 발생!

상수 값을 편하게 사용!