클래스 내부 구성요소 #1

필드/메서드/생성자

클래스 내부 구성요소 #1 - 필드

클래스 내부 구성요소 #1 - 필드

TIP
- 메서드 내부의 지역변수는 메서드가 호출 될 때에만 생성
- 메서드 실행 완료시 stack 메모리에서 삭제

☞ 필드(Field)

- 객체의 속성값을 지정할 수 있는 클래스에 포함된 변수
- **[비교]** 지역변수(local variable)는 메서드에 포함된 변수

1

필드 → Heap 메모리에 저장

지역변수 → Stack 메모리에 저장

필드
상위 중괄호가 클래스인 경우

지역변수
상위 중괄호가 메서드인 경우

2

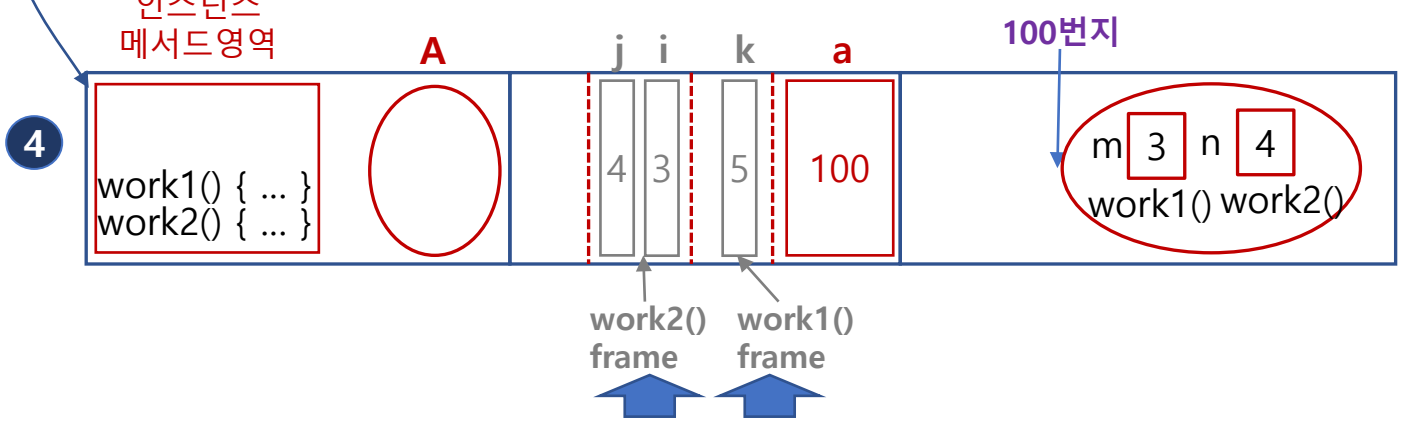
```
class A{

  int m=3;
  int n=4;

  void work1(){
    int k =5;
    System.out.println(k);
    work2(3);
  }

  void work2(int i){
    int j=4;
    System.out.println(i+j);
  }

}
```

3

```
A a = new A();
System.out.println(a.m);
System.out.println(a.n);
a.work1();
```

```
3
4
5
7
```

4



각 메서드 실행이 끝나면 메모리에서 사라짐

클래스 내부 구성요소 #1 - 필드

☞ 필드(Field) vs. 지역변수(local variable)

2 **필드** → Heap 메모리에 저장

지역변수 → Stack 메모리에 저장

- 필드와 지역변수의 초기값

1
- Heap 메모리에 들어가는 필드값은 값을 **미입력시 강제 초기화**됨
- Stack 메모리에 들어가는 지역변수는 강제 **초기화 되지 않음**

초기화 하지 않음

3
```
class A{

  int m;
  int n;

  void work1(){
    int k;
    System.out.println(k); //오류
  }

}
```

4
```
A a = new A();
System.out.println(a.m);
System.out.println(a.n);
//a.work1();
```

```
0
0
```

초기값이 없는 상태로 출력을 시도하여 오류 발생

클래스 내부 구성요소 #1 - 필드

☞ 필드(Field) vs. 지역변수(local variable)

필드 → Heap 메모리에 저장

지역변수 → Stack 메모리에 저장

- 필드와 지역변수의 초기값

초기화 하지 않음

```
class A {

  boolean m1;
  int m2;
  double m3;
  String m4;

  void printFieldValues() {
    System.out.println(m1);
    System.out.println(m2);
    System.out.println(m3);
    System.out.println(m4);
  }
}
```

1

2

TIP

Heap 메모리의 초기값

- 빈칸으로 존재할 수 없으며 디폴트 초기값이 강제 설정
- 기본자료형
  숫자(int, double 등) 디폴트값 : 0
  boolean 디폴트: false
- 참조자료형 디폴트값: null

3

```
A a = new A();
a.printFieldValues();
```

```
false
0
0.0
null
```

The End

클래스 내부 구성요소 #1 – 메서드

클래스 내부 구성요소 #1 – 메서드

☞ 메서드의 구조

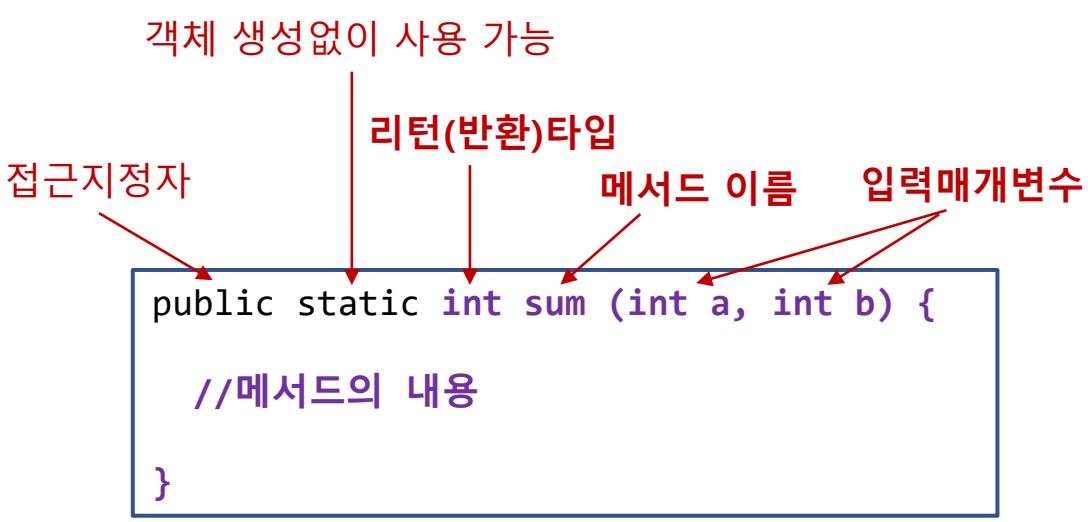
TIP

메서드의 장점
- 중복코드 재사용
- 코드의 모듈화를 통해 코드 가독성 향상

3

- **메서드의 정의** 생략 가능해도 적어주는 것이 좋다.
  생략가능한 것: 접근지정자, static, 매개변수

1



TIP

- 메서드 정의시 소괄호(( ))와 중괄호({ })가 포함

4

2

리턴(반환) 타입
- 메서드 완료 후 반환되는 타입
- 반환값이 있는 경우
  (=리턴타입이 void가 아닌 경우)
  메서드 내 return이 존재해야 함
- 반환값이 없는 메서드는
  리턴타입을 void로 선언

메서드 이름
- 변수의 이름 규칙과 동일하게 적용

  인터페이스는 이름 뒤에 able
  boolean 리턴 시 is로 시작

입력매개변수
- 메서드에 전달되는 값

클래스 내부 구성요소 #1 – 메서드

☞ 메서드의 예시

리턴타입 void + 매개변수 없음

1

```
void print () {
  System.out.println("안녕");
}
```

→ 리턴 타입이 void로 리턴하지 않는 메서드

리턴타입 int + 매개변수 없음

2

```
int data () {
  return 3;
}
```

→ 리턴 타입이 있는 경우 메서드 내에서는 return 포함
매개변수는 없음

리턴타입 double + 매개변수 2개

3

```
double sum (int a, double b) {
  return a+b;
}
```

→ 리턴 타입이 있는 경우 메서드 내에서는 return 포함
매개변수는 여러 개가 들어갈 수 있음

TIP

- 리턴타입이 void인 경우도 메서드 내에 return 사용가능
  단, 값의 리턴없이 return만 사용(메서드 종료)

4

```
void printMonth (int m) {
  if (m<0 || m>12){
    System.out.println("잘못된 입력!");
    return;
  }
  System.out.println(m+"월 입니다.");
}
```

클래스 내부 구성요소 #1 – 메서드

☞ 클래스 외부 메서드 호출

1

- **STEP #1. 객체 생성**
- **STEP #2. 참조변수로 부터 메서드 호출**

```java
public class A {
  //#. 리턴타입:void + 매개변수: 없음
  void print() {
    System.out.println("안녕");
  }
  //#. 리턴타입: int + 매개변수: 없음
  int data() {
    return 3;
  }
  //#. 리턴타입: double + 매개변수: 2개
  double sum(int a, double b) {
      return a+b;
  }
  //#. 리턴타입: void + 매개변수 : 1개 + 내부 함수종료(return 포함)
  void printMonth(int m) {
    if(m<1 || m>12) {
      System.out.println("잘못된 입력!");
      return;
    }
    System.out.println(m+"월 입니다.");
  }
}
```

2

```java
public class B {
  public static void main(String[] ar) {

    //#1. 객체 생성
    A a = new A();

    //#2. 참조변수로부터 메서드 호출
    a.print(); //"안녕"

    int k = a.data();
    System.out.println(k);//3

    double result = a.sum(3, 5.2);
    System.out.println(result);//8.2

    a.printMonth(5); //5월 입니다.
    a.printMonth(15);//잘못된 입력
  }
}
```

3

클래스 내부 구성요소 #1 – 메서드

TIP

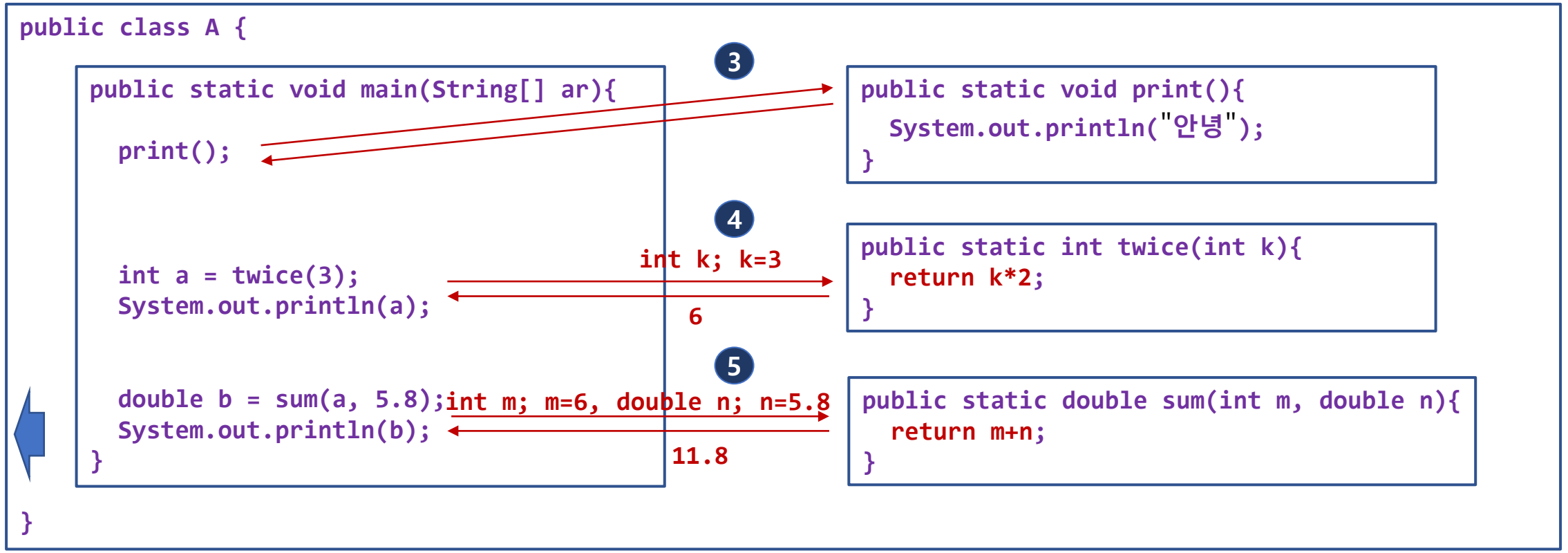
- 매개변수가 있는 메서드는 메서드 호출시 매개변수 선언 및 변수 값 대입을 가장 먼저 수행

TIP

- static 메서드에서는 static 메서드만 호출 가능

☞ 클래스 내부 메서드 호출

1 - **STEP #1. 메서드 이름으로 바로 호출**



클래스 내부 구성요소 #1 – 메서드

☞ 배열 매개변수를 가지는 메서드 호출

TIP

- 매개변수가 있는 메서드는 메서드 호출시 매개변수 선언 및 변수 값 대입을 가장 먼저 수행

TIP

- static 메서드에서는 static 메서드만 호출 가능

```
public static void main(String[] ar){


  printArray(new int[]{1,2,3});


  printArray({1,2,3});


}
```

1

```
int[] a;
a = new int[]{1,2,3}
```

2

```
int[] a = new int[]{1,2,3}; (O)
int[] a;
a = new int[]{1,2,3} (O)
```

```
public static void printArray(int[] a){

  System.out.println(Arrays.toString(a));

}
```

3

```
int[] a;
a = {1,2,3}
```

오류: 배열의 선언/초기화 참조

4

```
int[] a;
a = {1,2,3} (X)
```

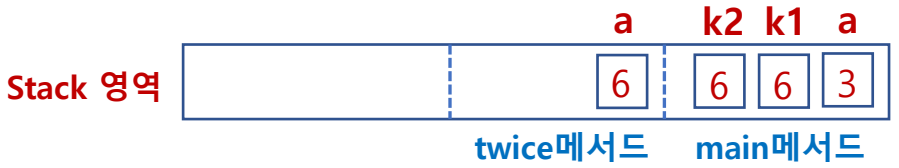
[1, 2, 3]

클래스 내부 구성요소 #1 – 메서드

1 ☞ **기본자료형** 매개변수 값 변화

TIP

2 - 지역변수는 메서드별로 별도의 공간에 만들어짐
- 매개변수를 전달하면 변수값만 복사되어 전달




```
public static void main(String[] ar){

  int a=3;


  int k1=twice(3);


  int k2=twice(a);


  System.out.println(k1);
  System.out.println(k2);
  System.out.println(a);
}
```

3 int a; a=3;
6

4 int a; a=3;
6

```
public static int twice(int a){
  a=a*2;
  return a;
}
```

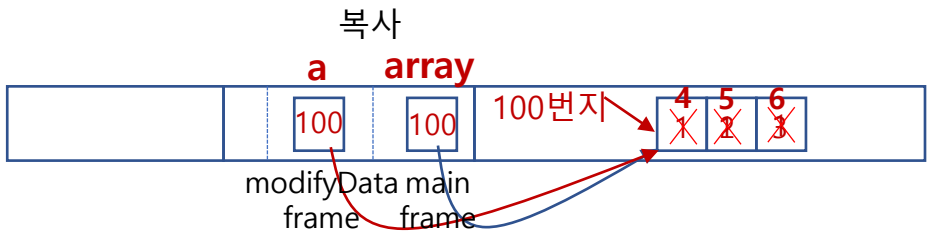
```
6
6
3
```

클래스 내부 구성요소 #1 – 메서드

1 ☞ 참조자료형 매개변수 값 변화

TIP

2 - 매개변수로 참조변수가 전달된 경우 번지(위치) 값이 전달되어 호출 메서드에서 값 변경시 최초 메서드에서 참조값도 함께 변경




```
public static void main(String[] ar){

  int[] array = new int[]{1,2,3};


  modifyData(array);


  printArray(array);


}
```

int[] a;
a = array

4

```
public static void modifyData(int[] a){
  a[0]=4; a[1]=5; a[2]=6;
  //printArray(a); //[4, 5, 6]
}
```

int[] a;
a = array

5

```
public static void printArray(int[] a){
  System.out.println(Arrays.toString(a));
}
```

```
[4, 5, 6]
```

TIP

6 Arrays.toString(1차원배열) → 배열값 출력 정적 메서드 ex) System.out.println(Arrays.toString(new int[]{1,2,3}); //[1,2,3]

클래스 내부 구성요소 #1 – 메서드

☞ 메서드 **오버로딩 (Overloading)**

TIP

2 메서드 시그너처 (메서드이름/매개변수 타입)
- public static int **sum** (**int** a, **int** b) { }

1
- **컴파일러는 메서드 시그너처(Method Signature)가 다르면 메서드 이름이 동일하여도 다른 메서드로 인식**

3

```
public static void main(String[] ar){
  print();
  print(3);
  print(5.8);
  print(2, 5);
}
```

```
public static void print(){
  System.out.println("데이터가 없습니다");
}
```

```
public static void print(int a){
  System.out.println(a);
}
```

```
public static void print(double a){
  System.out.println(a);
}
```

```
public static void print(int a, int b){
  System.out.println(a + ", "+b);
}
```

4

```
데이터가 없습니다
3
5.8
2, 5
```

클래스 내부 구성요소 #1 – 메서드

☞ 메서드 **오버로딩 (Overloading)**

1 - **System.out.println()이 다양한 타입을 출력할 수 있는 이유**
클래스

2

```
System.out.println(true);
System.out.println(3);
System.out.println(5.8);
```

```
true
3
5.8
```

void	**println**() Terminates the current line by writing the line separator string.
void	**println**(boolean x) Prints a boolean and then terminate the line.
void	**println**(char x) Prints a character and then terminate the line.
void	**println**(char[] x) Prints an array of characters and then terminate the line.
void	**println**(double x) Prints a double and then terminate the line.
void	**println**(float x) Prints a float and then terminate the line.
void	**println**(int x) Prints an integer and then terminate the line.
void	**println**(long x) Prints a long and then terminate the line.
void	**println**(**Object** x) Prints an Object and then terminate the line.
void	**println**(**String** x) Prints a String and then terminate the line.

3

클래스 내부 구성요소 #1 – 메서드

☞ 메서드 **오버로딩 (Overloading)**

1 - **리턴타입은 시그너처에 포함되지 않아 리턴 타입만 다른 경우 동일 메서드 2개가 정의된 것으로 보아 오류 발생**

```
public static void print(int a){
  …
}
```

```
public static int print(int a){
  …
}
```

2 **print(3) ??**

3 리턴타입은 시그너처에 포함되지 않음
이 두 메서드의 시그너처는 동일하여 오버로딩 불가

4 TIP

함수의 호출시에 리턴타입이 나타나지 않기 때문에
메서드 시그너처에는 리턴타입이 포함되지 않음

클래스 내부 구성요소 #1 – 메서드

1 ☞ 메서드의 **가변 길이 배열 매개변수** - **매서드의 매개변수로 전달된 원소의 개수만큼 배열을 생성하여 사용**

2 **가변 길이 배열 매개변수 문법**

```
리턴타입 메서드이름 (자료형... 참조변수){
  …
}
```

3

```
public static void method1(int... values) {
  System.out.println("매개변수 길이 : "+values.length);
  for(int i=0; i<values.length; i++)
    System.out.print+(values[i]+" ");
  System.out.println();
}
```

4

```
public static void method2(String...values) {
  System.out.println("매개변수 길이 : "+values.length);
  for(int i=0; i<values.length; i++)
    System.out.print(values[i]+" ");
  System.out.println();
}
```

5

```
public static void main(String[] args) {
  //#1. 길이가 정해져 있지 않은 int배열 매개변수
  method1(1,2); //매개변수길이:2 --> 1,2
  method1(1,2,3); //매개변수길이:3 --> 1,2,3
  method1(); //매개변수길이:0 --> X

  //#2. 길이가 정해져 있지 않은 String배열 매개변수
  method2("안녕","방가"); //매개변수길이:2 --> "안녕","방가"
  method2("땡큐","베리","감사"); //매개변수길이:3 --> "땡큐","베리","감사"
  method2(); //매개변수길이:0 --> X
}
```

```
매개변수 길이 : 2
1 2
매개변수 길이 : 3
1 2 3
매개변수 길이 : 0

매개변수 길이 : 2
안녕 방가
매개변수 길이 : 3
땡큐 베리 감사
매개변수 길이 : 0
```

The End

클래스 내부 구성요소 #1– 생성자

클래스 내부 구성요소 #1 – 생성자

☞ 클래스의 **생성자**

- 생성자의 2가지 특징

1. - 클래스의 이름과 동일
   - 반환(리턴)타입이 존재하지 않음

- 생성자의 주요 역할

4. - 객체 생성 및 필드 초기화

- 생성자를 정의하지 않은 경우 → **기본 생성자의 자동 추가**

```
class A{

  int m;
  void work(){
    …
  }

}
```

5. **생성자를 정의하지 않는 경우 컴파일러가 기본생성자 추가**

```
class A{

  int m;
  void work(){
    …
  }
  A() {

  }

}
```

정의

기본생성자
- 입력매개변수가 없는 생성자

2. **정의**
   - 반환(리턴)하지 않는다(void) ≠ 반환(리턴)타입이 없다(생성자)

생성자가 없는 클래스 = **붕어빵을 찍을 수 없는 붕어빵 기계**

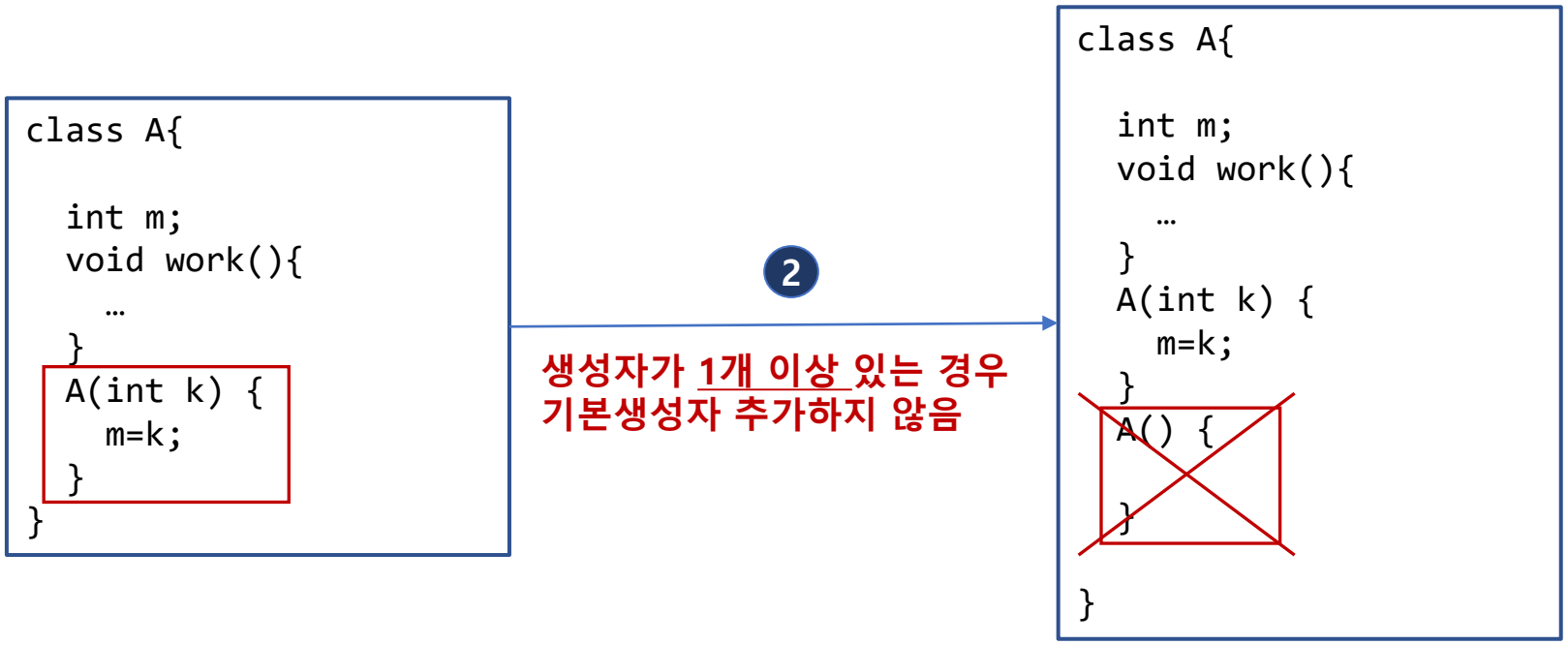
존재이유 없음

3. **모든 클래스는 생성자를 포함**

클래스 내부 구성요소 #1 – 생성자

☞ 클래스의 **생성자**

1 - 생성자를 정의한 경우 → 기본생성자를 추가 하지 않음



클래스 내부 구성요소 #1 – 생성자

☞ 클래스의 **생성자**

```
public static void main(String[] ar) {

  //#1. 기본생성자를 이용한 객체 생성
  A a = new A();
  B b = new B();
  //C c = new C(); //오류 (기본생성자 없음)
  C c = new C(3);

  //#2. 메서드 호출
  a.work();//0
  b.work();//0
  c.work();//3

}
```

2

```
class A {
  int m;
  void work() {
    System.out.println(m);
  }
  //기본생성자 자동 추가 A(){}
}
class B{
  int m;
  void work() {
    System.out.println(m);
  }
  B(){ //기본생성자
  }
}
class C{
  int m;
  void work() {
    System.out.println(m);
  }
  C(int a){
    m=a;
  }
}
```

1

생성자 내부의 내용은
생성자를 통해 **객체 생성** 이후의
추가작업 (주로 필드 초기화)

클래스 내부 구성요소 #1 – 생성자

☞ 클래스의 **생성자**

1 - 생성자의 오버로딩 → 생성자도 오버로딩(Overloading 가능)

생성자가 3개 → 객체생성방법 3개

```
class A{

  A() {
    System.out.println("첫번째 생성자");
  }

  A(int a) {
    System.out.println("두번째 생성자");
  }

  A(int a, int b) {
    System.out.println("세번째 생성자");
  }
}
```

2

객체 생성 방법

```
A a1 = new A();
```

```
A a2 = new A(3);
```

```
A a3 = new A(3, 5);
```

this 키워드와 this() 메서드

this 키워드와 this() 메서드

☞ this 키워드 vs. this() 메서드

- this 키워드 → **자신이 속한 클래스의 객체**

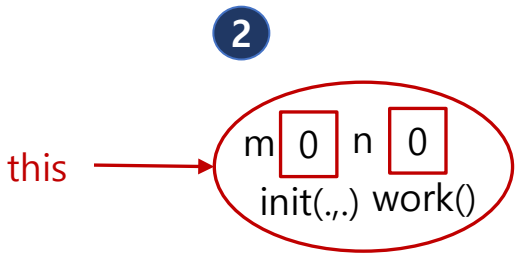
Tip ③

- static 메서드 내에서는 this 키워드를 사용할 수 없음
- 지역변수에는 this.가 붙지 않음

①
- 모든 필드와 메서드 활용시에는 소속과 함께 표기해야 함 (정의식에서는 사용하지 않음)
- 클래스 내부의 필드와 메서드에 소속을 표기하지 않는 경우 컴파일러가 자동으로 소속(this.)을 붙여 줌

```
class A {

  int m;
  int n;

  void init(int a, int b){
    int c=3;
    m=a;
    n=b;
  }
  void work(){
    init(2, 3);
  }

}
```

②



모든 필드와 메서드 활용시
컴파일러가
this.을 자동으로 붙임

```
class A {

  int m;
  int n;

  void init(int a, int b){
    int c=3;
    this.m=a;
    this.n=b;
  }
  void work(){
    this.init(2, 3);
  }

}
```

this 키워드와 this() 메서드

☞ this 키워드 vs. this() 메서드

- this 키워드 → **자신이 속한 클래스의 객체**

Tip

1
- **필드**는 클래스 내부 전체 영역에서 사용 가능
- **지역변수**는 해당 메서드 내부에서만 사용 가능
- **필드와 지역변수 모두를 사용가능한 영역에서는 지역변수로 인식**

- 모든 필드와 메서드 활용시에는 소속과 함께 표기해야 함 (정의식에서는 사용하지 않음)
- 클래스 내부의 필드와 메서드에 소속을 표기하지 않는 경우 컴파일러가 자동으로 소속(this.)을 붙여 줌

비추!! 오류 발생하기 좋음

```
class A{

  int m;
  int n;

  void init(int m, int n){
    m=m;
    n=n;
  }

}
```

필드와 지역변수를 모두 사용할 수 있고 이름이 같은 경우 지역변수로 인식

2

```
A a = new A();
a.init(3,4);
System.out.println(a.m);
System.out.println(a.n);
```

```
0
0
```

3

```
class A{

  int m;
  int n;

  void init(int m, int n){
    this.m=m;
    this.n=n;
  }

}
```

```
A a = new A();
a.init(3,4);
System.out.println(a.m);
System.out.println(a.n);
```

```
3
4
```

this 키워드와 this() 메서드

☞ this 키워드 vs. this() 메서드

- this() 메서드 → **자기 클래스 내부의 다른 생성자를 호출**

1

- this() 메서드는 **생성자 내부에서만 사용**가능
- 반드시 중괄호 이후 **첫 줄에 위치**하여야 함

```
class A{

  A(){
    System.out.println("첫번째 생성자");
  }

  A(int a){
    this(); //반드시 첫 줄에 위치
    System.out.println("두번째 생성자");
  }

}
```

2

3

```
A a1 = new A();
```

→ 첫번째 생성자

```
A a2 = new A(3);
```

→ 첫번째 생성자
두번째 생성자

this 키워드와 this() 메서드

1

☞ this 키워드 vs. this() 메서드 - this() 메서드의 **활용 예** → 생성자에서 여러 개의 변수를 초기화 하는 경우

```
class A{
  int m1, m2, m3, m4;
  A(){
    m1=1;
    m2=2;
    m3=3;
    m4=4;
  }
  A(int a){
    m1=a;
    m2=2;
    m3=3;
    m4=4;
  }
  A(int a, int b){
    m1=a;
    m2=b;
    m3=3;
    m4=4;
  }
}
```

2

this() 메서드를 이용하여 생성자의 중복성 제거

```
class A{
  int m1, m2, m3, m4;
  A(){
    m1=1;
    m2=2;
    m3=3;
    m4=4;
  }
  A(int a){
    this();
    m1=a;
  }
  A(int a, int b){
    this(a);
    m2=b;
  }
}
```

The End