객체지향 프로그래밍

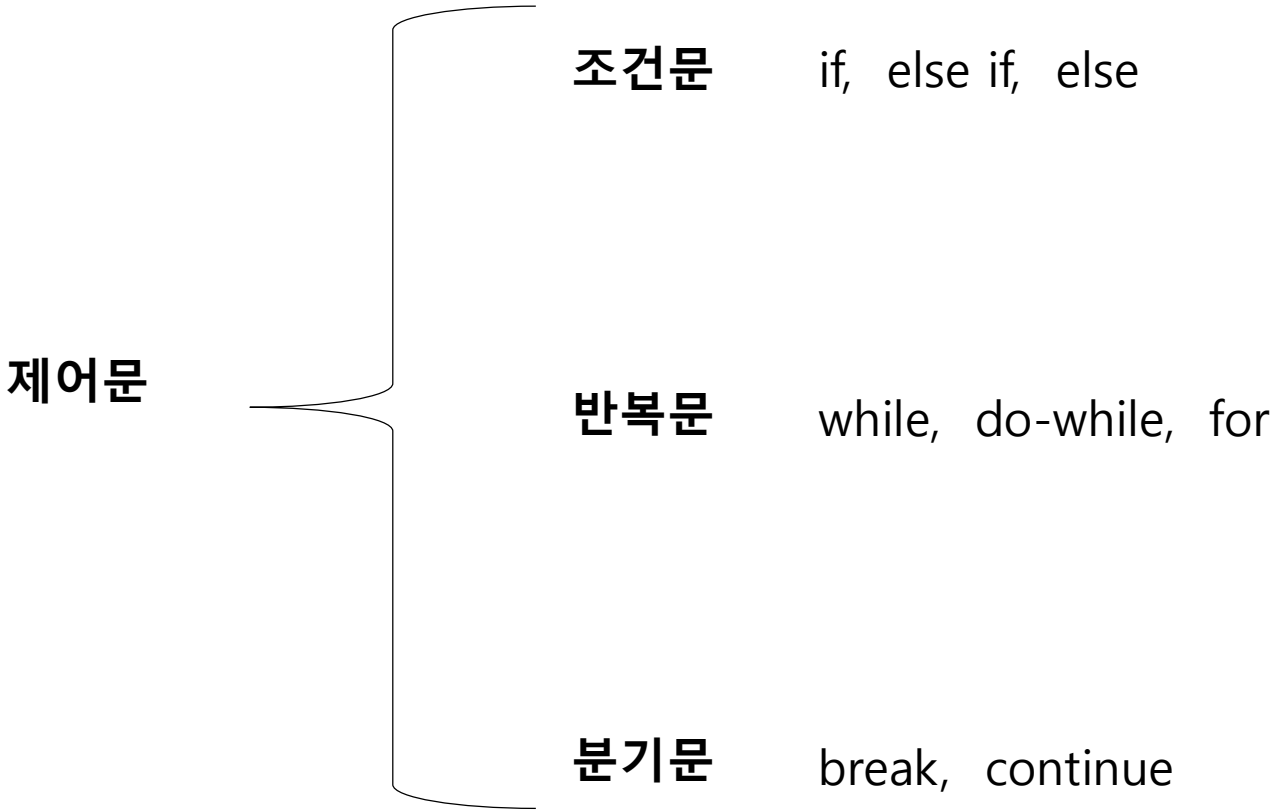
Object-Oriented Programming

Chapter 03. 제어문과 메서드

1. 제어문
    1. 조건문
    2. 반복문
    3. 분기문
2. 메서드

Recap

제어문 - Recap



switch 문

switch 문

- **조건문**의 일종

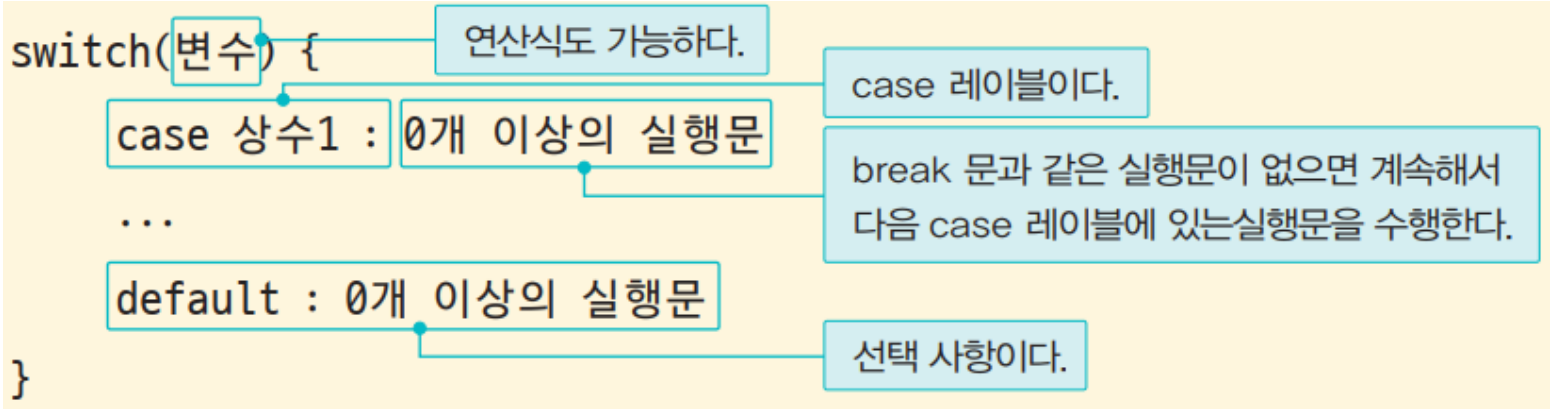
- **두 버전의 구문 존재**
  - 기존의 낙하(fall-through) 방식
  - Java 14에서 개선된 방식

- Java는 **하위호환성**(backward compatibility)을 보장하므로,
  Java 14 이후 버전에서도 낙하 방식 사용 가능

switch 문

switch 문 – 낙하(fall-through) 방식

- **구성요소**
  - switch 변수 (데이터 타입은 정수, 열거형, 문자열로 한정)
  - case 레이블
  - 실행문
  - default



switch 문

switch 문 – 낙하(fall-through) 방식

example

```
int month = 8;
String monthString;
switch (month) {
  case 1:
    monthString = "January";
    break;
  case 2:
    monthString = "February";
    break;
  .
  .
  .
  default: monthString = "invalid";
           break;
}
System.out.println(monthString);
```

switch 문

Practice

- **예제 3-15**: 문자열 타입 switch 문 (p.103)
  - package: chap03
  - class: Switch2Demo

switch 문

switch 문 – 개선된 방식

- **도입된 요소**
  - 다중 case 레이블
  - 화살표 case 레이블
  - yield 예약어

- 위 요소들을 도입해 가독성을 높이고 오류를 방지

- Java 14 이상의 버전에서 사용 가능

switch 문

switch 문 – 개선된 방식

다중 case 레이블

- 콤마로 연결된 다수의 상수를 case 레이블로 사용 가능

```
String day = "SATURDAY";
String typeOfDay;

switch (day) {
    case "MONDAY", "TUESDAY", "WEDNESDAY", "THURSDAY", "FRIDAY":
        typeOfDay = "Weekday";
        break;
    case "SATURDAY", "SUNDAY":
        typeOfDay = "Weekend";
        break;
    default:
        typeOfDay = "invalid";
}

System.out.println(typeOfDay);
```

switch 문

switch 문 – 개선된 방식

화살표 연산자

- 콤마로 연결된 다수의 상수를 case 레이블로 사용 가능
- switch 문에 대한 반환값 도입
- switch 변수의 모든 가능한 값(value)에 대해 case 레이블을 사용하거나, default를 사용해야 함

```
String day = "SATURDAY";

String typeOfDay = switch (day) {
    case "MONDAY", "TUESDAY", "WEDNESDAY", "THURSDAY", "FRIDAY" -> "Weekday";
    case "SATURDAY", "SUNDAY" -> "Weekend";
    default -> "invalid"
};

System.out.println(typeOfDay);
```

switch 문

switch 문 – 개선된 방식

화살표 연산자

- 여러 개의 실행문을 사용하기 위해서는 중괄호로 감싼 실행문 블록 사용

```
String day = "MONDAY";
String typeOfDay = "";

switch (day) {
    case "MONDAY", "TUESDAY", "WEDNESDAY",
          "THURSDAY", "FRIDAY" -> typeOfDay = "Weekday";

    case "SATURDAY", "SUNDAY" -> {
        System.out.println("Weekend processing...");
        typeOfDay = "Weekend";
    }

    default -> {
        System.out.println("Unknown day");
        typeOfDay = "Unknown";
    }
}

System.out.println(typeOfDay);
```

switch 문

Practice

- **예제 3-16**: 개선된 switch 문 (p.105)
  - package: chap03
  - class: Switch3Demo

switch 문

switch 문 – 개선된 방식

yield 예약어

- 값을 반환하면서 switch 연산식을 종료

```
String day = "SATURDAY";

String typeOfDay = switch (day) {
    case "MONDAY" -> "Weekday";
    case "SATURDAY" -> {
        System.out.println("Weekend processing...");
        yield "Weekend";
    }
    default -> {
        System.out.println("Unknown day");
        yield "Unknown";
    }
};

System.out.println(typeOfDay);
```

메서드

메서드(method)

- **특정 연산을 수행하기 위해 실행문을 모아 둔 블록**
- 함수와 유사
- 클래스 내부에서만 선언 가능

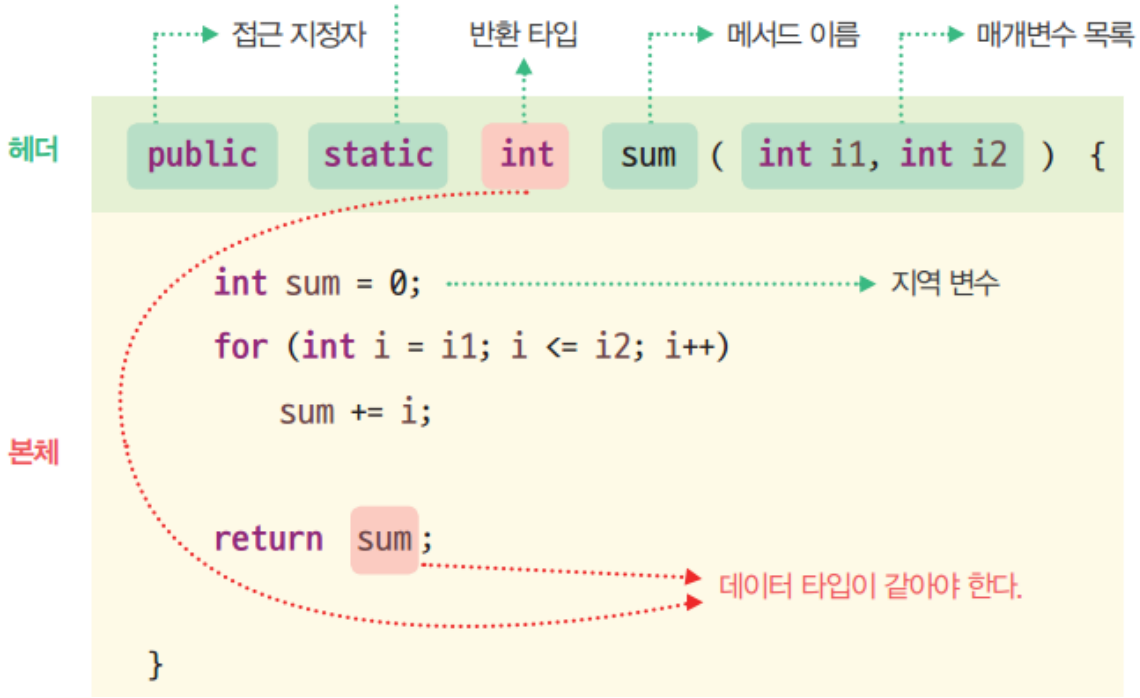
메서드 사용의 이점

- 중복 코드를 줄이고 코드를 재사용할 수 있다.
- 코드를 모듈화해 가독성을 높이므로 프로그램의 품질을 향상시킨다.

메서드

메서드의 구조

- **헤더(header):** 선언할 메서드의 기본적인 특징들 표현
- **본체(body):** 실행할 코드 나열



메서드

메서드의 주요 구성 요소

- 이름
- 매개변수
- 반환 타입
- 본체

메서드

메서드의 주요 구성 요소

이름

- 변수, 클래스와 같이 메서드 또한 이름을 가진다.

- 메서드의 기능과 목적을 명확히 나타내는 이름을 설정해 코드 가독성을 높일 수 있다.

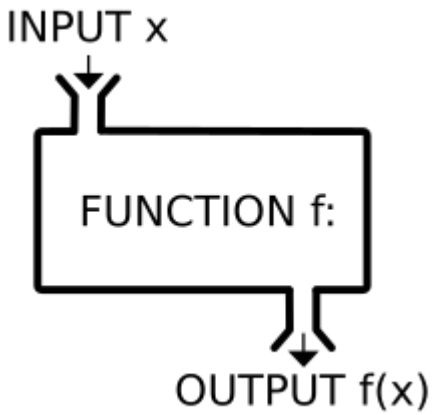
```java
public class Switch3Demo {

    public static void main(String[] args) {
        whoIsIt("호랑이");
        whoIsIt("참새");
        whoIsIt("고등어");
        whoIsIt("곰팡이");
    }

    static void whoIsIt(String bio) {
        String kind = "...";
        switch (bio) {
            case "호랑이", "사자" -> kind = "포유류";
            case "독수리", "참새" -> kind = "조류";
            case "고등어", "연어" -> kind = "어류";
            default -> System.out.print("어이쿠! ");
        }
        System.out.printf("%s는 %s이다.\n", bio, kind);
    }
```

메서드

메서드의 주요 구성 요소

매개변수 (parameter)

- 메서드는 함수와 유사한 성질을 지닌다.
- 입력과 출력



메서드

메서드의 주요 구성 요소

매개변수 (parameter)

- 메서드 실행 시 매개변수를 통해 지정된 형태로 데이터들을 입력 받고, 본체(body)에서는 입력 받은 데이터들을 활용할 수 있다.

```java
public static void add(int x, int y) {
    System.out.println(x + y);
}
```

메서드

메서드의 주요 구성 요소

매개변수의 선언

- 메서드 헤더에 매개변수의 타입과 이름을 나열

```java
public static void printScore(String name, int score) {
    System.out.println(name + " 학생의 점수: " + score);
}
```

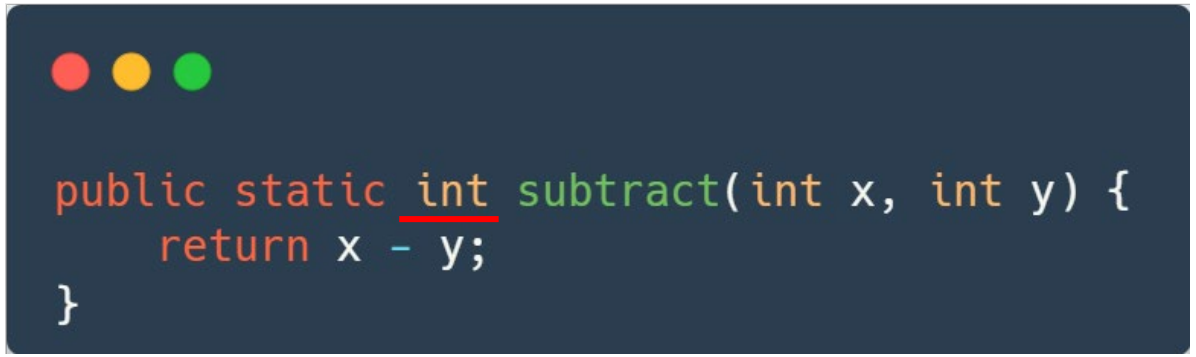
```java
public static void add(int x, int y) {
    System.out.println(x + y);
}
```

메서드

메서드의 주요 구성 요소

반환 타입

- 메서드는 값을 반환할 수 있다.
- 반환 타입은 메서드가 본체를 실행 한 후 반환 할 데이터 타입이다 .
- 반환할 데이터가 없다면 반환 타입으로 **void**를 사용한다.



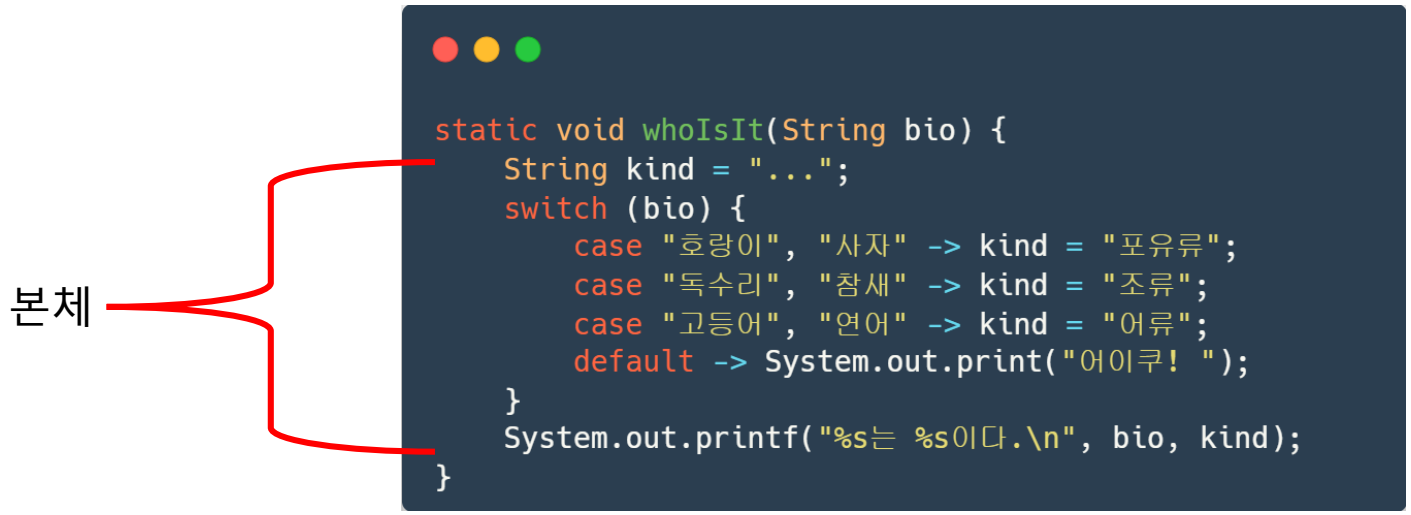
```
public static int subtract(int x, int y) {
    return x - y;
}
```

메서드

메서드의 주요 구성 요소

본체(body)

- 메서드는 실행의 대상이다.
- 메서드 호출 시 실행될 실행문들을 중괄호로 감싸 본체에 작성한다.



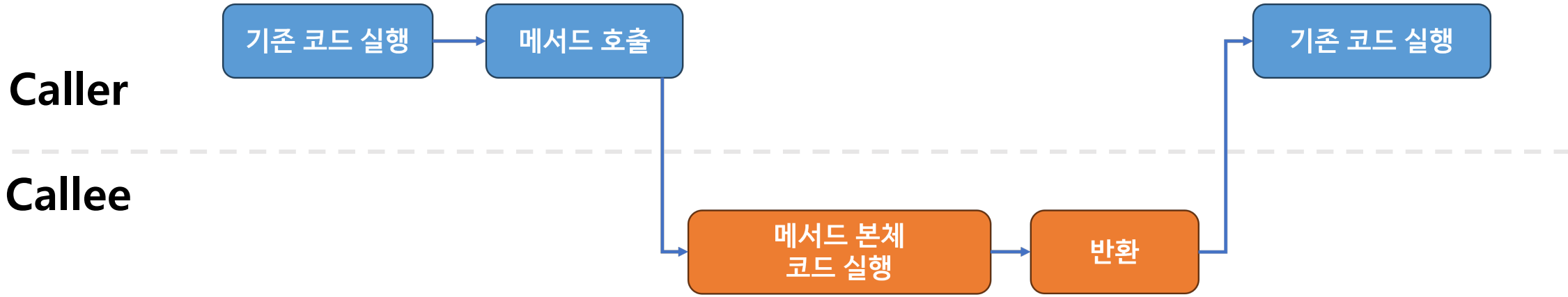
메서드

메서드의 호출과 반환

실행 흐름 변화

메서드 호출 시 제어는 호출된 메서드로 넘어간 후 다시 호출한 메서드로 돌아온다.

- **Callee**: 호출된 메서드
- **Caller**: callee를 호출한 메서드



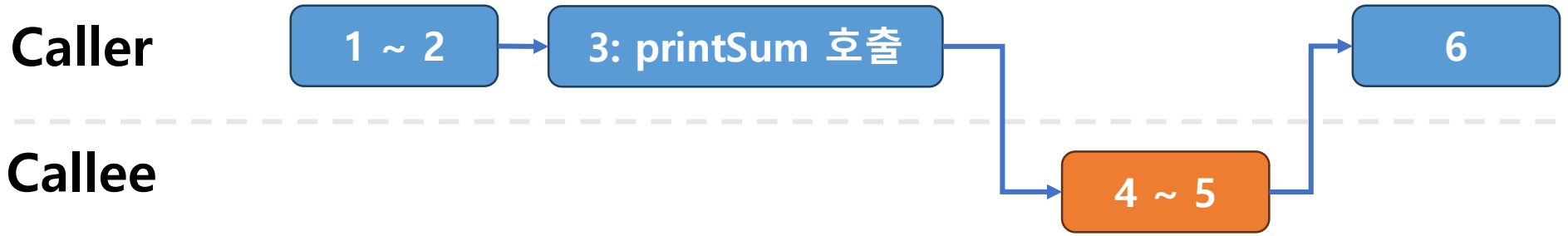
메서드

메서드의 호출과 반환

예시

- **Callee**: printSum
- **Caller**: main

```
 public static void main(String[] args) {
1  int num1 = 10, num2 = 12;
2  System.out.println(num1 + " + " + num2 + "은 ");
3  printSum(num1, num2);
6  System.out.println("입니다.");
 }

 public static void printSum(int a, int b) {
4  int c = a + b;
5  System.out.println(c);
 }
```

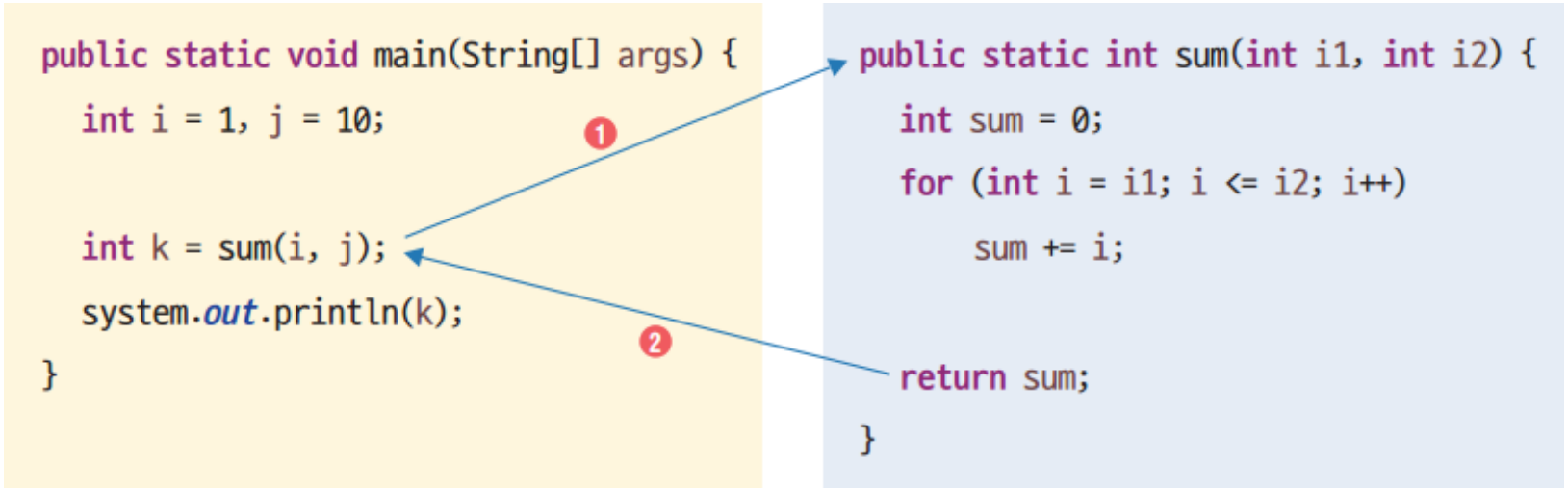


메서드

메서드의 호출과 반환

return 문

- 값을 반환하면서 메서드의 실행을 종료



메서드

메서드의 호출과 반환

return 문

- 값을 반환하면서 메서드의 실행을 종료

반환 값이 없는 void 타입 메서드에서 return 문

```
public static void printScore(int score) {
    if (score < 0 || score > 100) {
        System.out.println("잘못된 점수 : " + score);
        return;
    }
    System.out.println("점수 : " + score);
}
```

메서드

메서드의 호출과 반환

return 문

- 값을 반환하면서 메서드의 실행을 종료

```java
public static String checkNumber(int num) {
    if (num > 0) {
        return "양수입니다.";
    }
    if (num < 0) {
        return "음수입니다.";
    }
    return "0입니다.";
}
```

메서드

Practice

- **예제 3-20**: 메서드 활용 구간 누적합 (p.109)
  - package: chap03
  - class: Method2Demo

- **예제 3-21**: 메서드에서 return 활용 (p.112)
  - package: chap03
  - class: ReturnDemo

메서드

매개변수와 값 전달

- 메서드를 호출하려면 매개변수에 대응되는 값을 제공해야 한다.
- **인수(argument):** 메서드 호출 시 호출측(caller)에서 제공하는 매개변수에 대응되는 값
- 매개변수와 인수의 순서, 타입, 개수가 일치해야 한다.

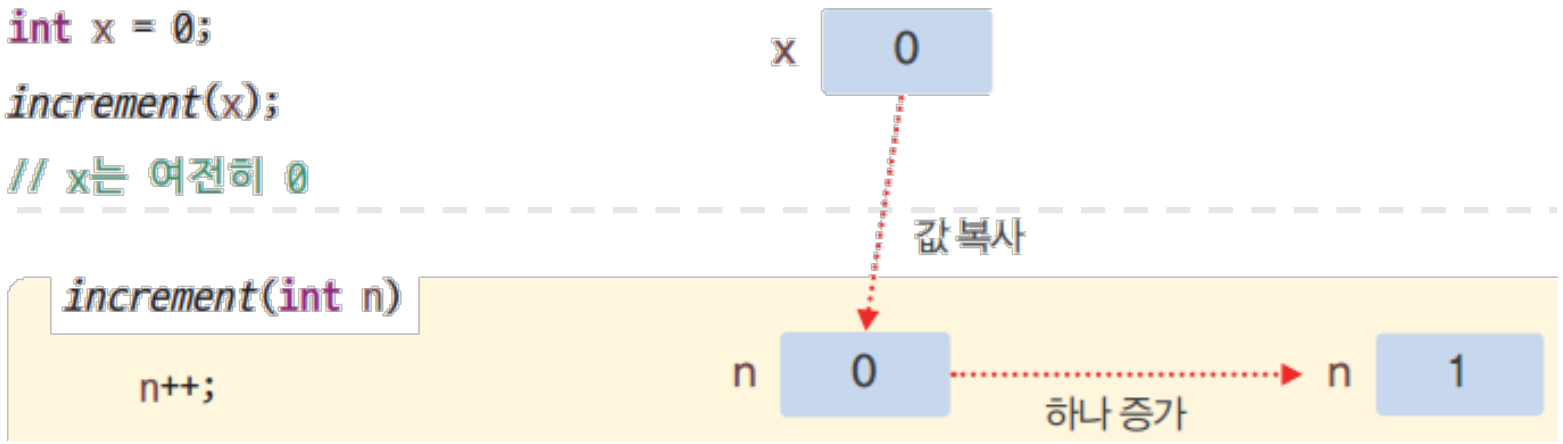
```java
public class EchoDemo {
    public static void main(String[] args) {
        echo("안녕!", 3);
    }
      (Argument) String, int

    public static void echo(String s, int n) {
        for (int i = 0; i < n; i++)
            System.out.println(s);
    }
}
```

(Parameter) String, int

메서드

값 전달(call by value)

- 메서드를 호출하면 인숫값의 **복사본**을 매개변수로 전달한다.
- 메서드 실행 도중 매개변수 값이 변해도 인수에는 영향을 미치지 않는다.



메서드

Practice

- **예제 3-23**: 메서드와 값 전달 (p.114)
  - package: chap03
  - class: IncrementDemo

메서드

메서드 시그니처 (method signature)

메서드의 이름, 매개변수의 개수, 매개변수의 데이터 타입, 매개변수의 순서

Example:

- echo(String, int)
- main(String[ ])
- sum(int, int)

메서드

메서드 시그니처 (method signature)

- Java에서는 두 메서드의 이름이 동일하더라도 시그니처가 다르면 서로 다른 메서드로 취급

Example

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

static int sum(int a, int b, int c) {
  return a + b + c;
}

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

메서드

메서드 오버로딩 (method overloading)

- **이름은 같지만 시그니처가 다른 메서드를 정의하는 것**

Example: 이름은 sum으로 동일하지만, 시그니처가 서로 다른 메서드들

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

static int sum(int a, int b, int c) {
  return a + b + c;
}

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

메서드

Practice

- **예제 3-24**: 메서드 오버로딩 (p.115)
  - package: chap03
  - class: OverloadDemo

Recap

Method

1. 구조

- Header
- Body

2. 구성 요소

- 이름
- 매개변수
- 반환 타입
- 실행문(본체)

3. 호출과 반환

- Caller
- Callee
- 실행 흐름 변화
- 값 전달(call by value)
- 인수 (argument)

4. 오버로딩

- 시그니처
  - 이름
  - 매개변수 타입 순서

Q & A

메서드

Challenge (메서드 기초)

- **도전 과제 2:** 팩토리얼 (p.118)
  - int 타입으로 자연수 x를 입력 받고, factorial(x)를 반환하는 메서드 구현
  - (단, 메서드의 이름은 factorial로 지정)

```java
static int factorial(int x) {
    int r = 1;
    // 팩토리얼 값을 계산하는 코드
    return r;
}
```

메서드

Challenge (메서드 오버로딩)

- **도전 과제 3:** – 구간 팩토리얼 (p.119)
  - int 타입으로 자연수 x, y를 입력 받고,

    factorial(y) / factorial(x) 을 반환하는 메서드 구현

  - (단, 메서드의 이름은 factorial로 지정, 도전과제 2의 클래스에 작성)

```
static int factorial(int x, int y) {
    int r = 1;
    // 팩토리얼 값을 계산하는 코드
    return r;
}
```

메서드

Practice (메서드 오버로딩)

다음은 foo() 메서드가 누락된 프로그램 일부와 실행 결과이다. foo() 메서드를 완성하라.

```
public static void main(String[] args) {
    foo("안녕", 1);
    foo("안녕하세요", 1, 2);
    foo("잘 있어");
}
```

```
안녕 1
안녕하세요 1 2
잘 있어
```

메서드

Practice

성적 관리 프로그램

- 학생 **3명**의 이름과 영어, 수학 점수를 입력받는다.
- 각 학생에 대해 **평균, 총점**을 출력한다.
- **점수 순위**를 1위부터 3위까지 내림차순으로 출력한다.

```
출력:  학생 1의 이름을 입력하세요.
입력:  홍길동
출력:  학생 1의 영어 점수를 입력하세요.
입력:  95
출력:  학생 1의 수학 점수를 입력하세요.
입력:  98
출력:  학생 2의 이름을 입력하세요.
. . .
출력:  홍길동 – 평균 96.5, 총점 193
출력:  이순신 – 평균 100, 총점 200
출력:  아무개 – 평균 50, 총점 100

출력:  순위 – 1. 이순신,  2. 홍길동,  3. 아무개
```