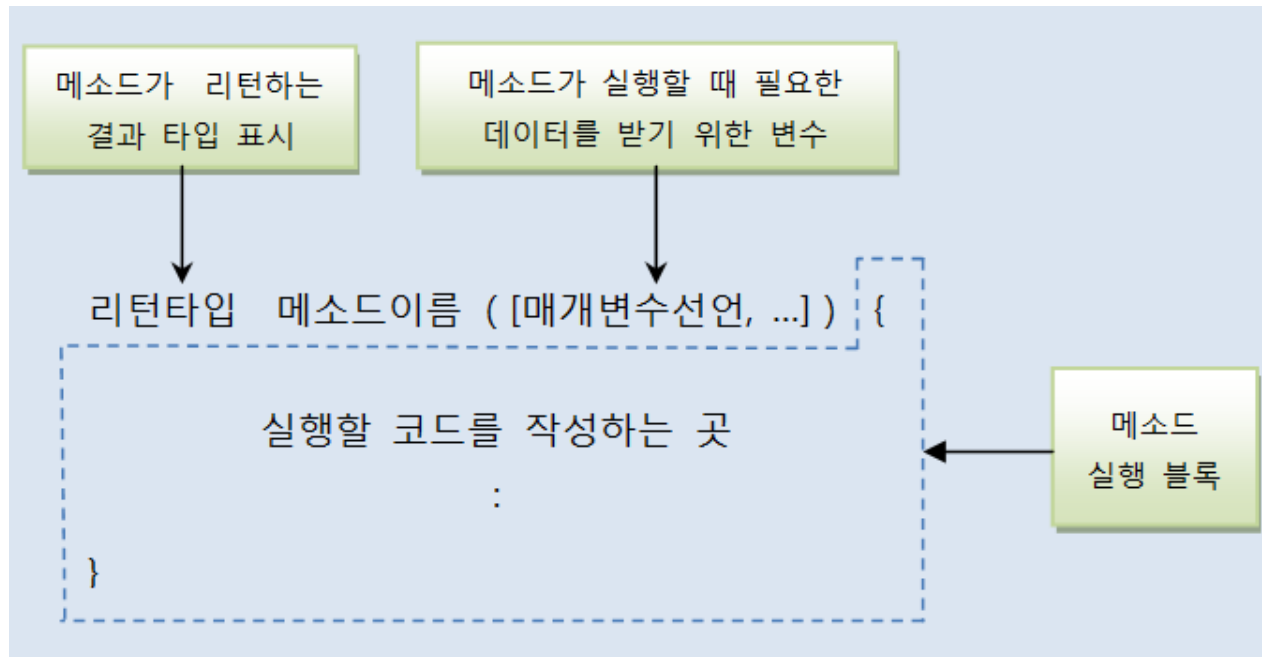


■ 메소드

- 어떤 작업을 수행하기 위한 명령문의 집합
- 주로 어떤 값을 입력받아서 처리한 후 결과를 되돌려 준다
- 반복적으로 사용되는 코드를 줄이기 위해서 사용
- 하나의 메소드는 한가지 기능만 수행하도록 작성하는 것이 좋다



■ 여러가지 형태의 메소드 (반환값 X, 매개변수 X)

```
public class MethodExam {  
    public static void main(String[] args) {  
        nothing(); /* 메소드 호출 */  
  
        int result = nothing(); /* 사용불가 - 반환값 */  
        nothing("100"); /* 사용불가 - 매개변수 */  
    }  
  
    public static void nothing() {  
        System.out.println("메소드 실행");  
    }  
}
```

■ 여러가지 형태의 메소드 (반환값 O, 매개변수 X)

```
public class MethodExam {  
    public static void main(String[] args) {  
        returnValue(); /* 메소드만 호출 */  
        int value = returnValue(); /* 메소드 호출 후 반환되는 값 변수로 대입 */  
  
        String result = returnValue(); /* 사용불가 - 변수타입 */  
        returnValue("100"); /* 사용불가 - 매개변수 */  
    }  
  
    public static int returnValue() {  
        System.out.println("메소드 실행");  
        return 100; /* 메소드의 반환타입과 반드시 같아야 됨 */  
    }  
}
```

■ 여러가지 형태의 메소드 (반환값 X, 매개변수 O)

```
public class MethodExam {  
    public static void main(String[] args) {  
        arguments(100, 200); /* 메소드 호출 */  
  
        String result = arguments(100, 200); /* 사용불가 - 반환값 */  
        arguments("100"); /* 사용불가 - 매개변수 */  
        arguments(100); /* 사용불가 - 매개변수 */  
        arguments(100, 200, 300); /* 사용불가 - 매개변수 */  
    }  
  
    public static void arguments(int a, int b) {  
        System.out.println("메소드 실행");  
        System.out.println("a + b의 값은 " + (a + b));  
    }  
}
```

■ 여러가지 형태의 메소드 (반환값 O, 매개변수 O)

```
public class MethodExam {  
    public static void main(String[] args) {  
        findMaxValue(100, 200); /* 메소드만 호출 */  
        int value = findMaxValue(2324, 91); /* 메소드 호출 후 반환되는 값 변수로 대입 */  
  
        String result = findMaxValue(2324, 91); /* 사용불가 - 반환값 */  
        findMaxValue("100"); /* 사용불가 - 매개변수 */  
        findMaxValue(100); /* 사용불가 - 매개변수 */  
        findMaxValue(100, 200, 300); /* 사용불가 - 매개변수 */  
    }  
  
    public static int findMaxValue(int a, int b) {  
        if(a > b) {  
            return a; /* 메소드의 반환타입과 반드시 같아야 됨 */  
        } else {  
            return b; /* 메소드의 반환타입과 반드시 같아야 됨 */  
        }  
    }  
}
```

■ 메소드 사용 - 1 (반환값 X)

```
public class MethodExam1 {  
    public static void main(String[] args) {  
        printStar(5, '★');  
    }  
  
    public static void printStar(int count, char ch) {  
        for(int i = 1; i <= count; i++) {  
            for(int j = 1; j <= i; j++) {  
                System.out.print(ch);  
            }  
            System.out.println();  
        }  
    }  
}
```

■ 메소드 사용 – 2 (반환값 O)

```
public class MethodExam2 {  
    public static void main(String[] args) {  
        String result = printStar(5, '★');  
        System.out.println(result);  
    }  
  
    public static String printStar(int count, char ch) {  
        String star = "";  
        for(int i = 1; i <= count; i++) {  
            for(int j = 1; j <= i; j++) {  
                star = star + ch;  
            }  
            star = star + "\n";  
        }  
        return star;  
    }  
}
```

■ 메소드 사용 – 3 (지정된 숫자 사이의 난수 생성)

```
public class MethodExam3 {  
    public static void main(String[] args) {  
        int number = getRandomNumber(60, 100);  
        System.out.println(number);  
    }  
  
    public static int getRandomNumber(int startNum, int endNum) {  
        Random random = new Random();  
        int number = 0;  
        while(true) {  
            number = random.nextInt(endNum);  
            if(number >= startNum) {  
                break;  
            }  
        }  
        return number;  
    }  
}
```


■ 매개변수의 개수를 모르는 경우

● 매개변수를 배열 타입으로 선언

```
int sum1(int[] values) { }
```

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

```
int result = sum1(values);
```

```
int result = sum1(new int[] { 1, 2, 3, 4, 5 });
```

● 값의 목록만 넘겨주는 방식 (가변인자)

```
int sum2(int ... values) { }
```

```
int result = sum2(1, 2, 3);
```

```
int result = sum2(1, 2, 3, 4, 5);
```

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

```
int result = sum2(values);
```

```
int result = sum2(new int[] { 1, 2, 3, 4, 5 });
```

■ return 문

- 값 반환
- 메소드의 실행 종료

■ 반환값이 있는 메소드

- 반드시 리턴(return)문 사용해 반환값 지정

```
int plus(int x, int y) {  
    int result = x + y;  
    return result;  
}
```


return 문 뒤에 실행문 올 수 없음

```
boolean isLeftGas() {  
    if(gas==0) {  
        System.out.println("gas 가 없습니다.");  
        return false;  
    }  
    System.out.println("gas 가 있습니다.");  
    return true;  
}
```

■ 반환값이 없는 메소드

● 메소드 내의 명령문 실행 도중 강제 종료(반환)

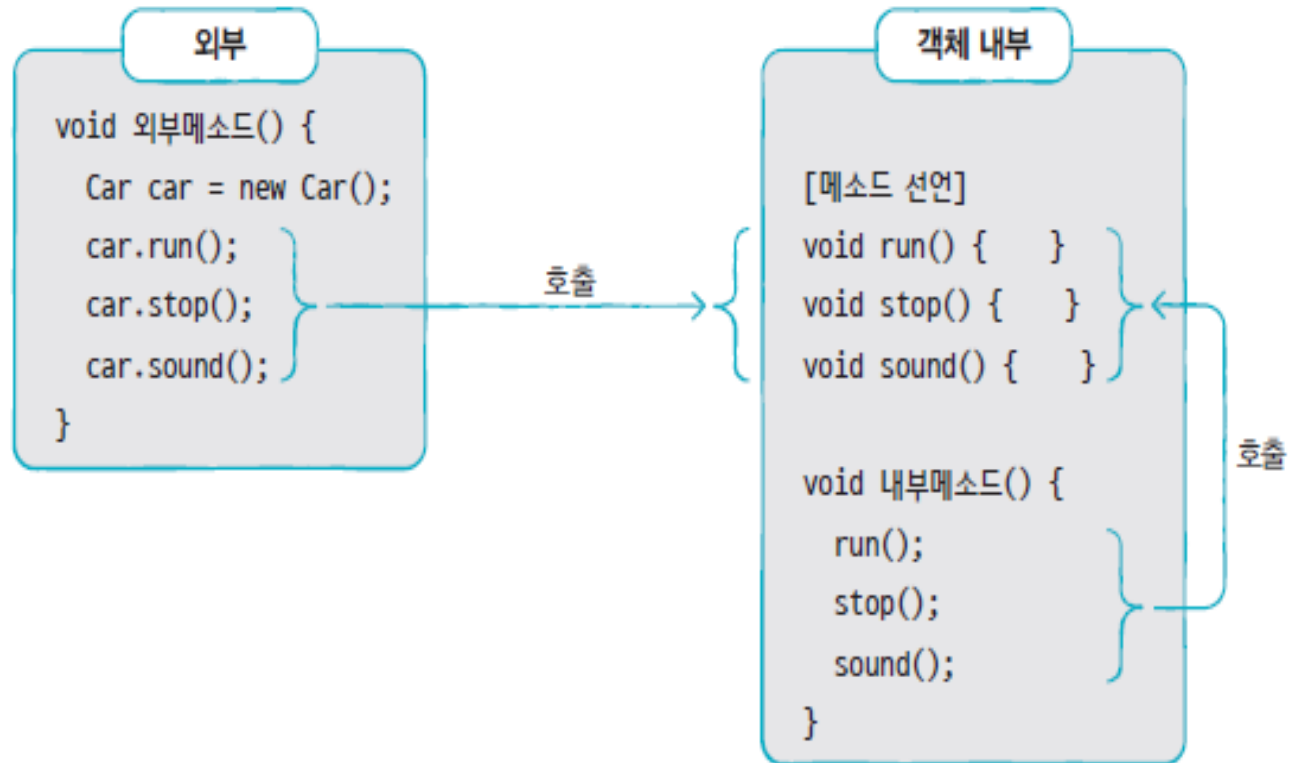
```
void run() {  
    while(true) {  
        if(gas > 0) {  
            System.out.println("달립니다.(gas잔량:" + gas + ")");  
            gas -= 1;  
        } else {  
            System.out.println("멈춥니다.(gas잔량:" + gas + ")");  
            return;  
        }  
    }  
}
```



run() 메소드 실행 종료

■ 메소드 호출

- 클래스 내부의 메소드는 이름만으로 호출
- 클래스 외부의 메소드는 객체를 생성한 뒤 참조 변수를 이용하여 호출



■ 메소드 호출

● 내부 메소드 사용

```
public class ClassName {  
    void method1( String p1, int p2 ) {  
          
    }  
  
    void method2() {  
        method1( "홍길동", 100 );  
    }  
}
```

The diagram illustrates the flow of a method call within a class. A blue arrow labeled ① 호출 (Call) originates from the `method1("홍길동", 100);` line inside `method2()` and points to the opening curly brace of `method1(String p1, int p2)`. Another blue arrow labeled ② 실행 (Execution) originates from the opening curly brace of `method1` and points downwards, indicating the execution of the method body.

■ 메소드 호출

● 외부 메소드 사용

1. 클래스를 객체로 생성 (메모리 등록)

```
클래스 참조변수 = new 클래스( 매개값, ... );
```

2. 참조변수와 Dot(.) 연산자를 사용하여 메소드 호출

```
참조변수.메소드( 매개값, ... ); //리턴값이 없거나, 있어도 리턴값을 받지 않을 경우  
타입 변수 = 참조변수.메소드( 매개값, ... ); //리턴값이 있고, 리턴값을 받고 싶을 경우
```

```
Car myCar = new Car();  
myCar.keyTurnOn();  
myCar.run();  
int speed = myCar.getSpeed();
```

■ 메소드 오버로딩

- 동일한 이름을 가지는 메소드를 만드는 행위
- 메소드로 넘겨주는 매개 값을 확인하여 메소드 호출

```
plus(10, 20);
```

```
int plus(int x, int y) {  
    int result = x + y;  
    return result;  
}
```

```
plus(10.5, 20.3);
```

```
double plus(double x, double y) {  
    double result = x + y;  
    return result;  
}
```

```
int x = 10;  
double y = 20.3;  
plus(x, y);
```

■ 메소드 오버로딩

- 매개변수의 타입, 개수, 순서 중 하나라도 달라야 오버로딩 성립

```
int divide(int x, int y) { ... }  
double divide(int boonja, int boonmo) { ... } (X)
```

- 오버로딩의 활용 예

- System.out.println()

```
void println() { ... }  
void println(boolean x) { ... }  
void println(char x) { ... }  
void println(char[] x) { ... }  
void println(double x) { ... }  
  
void println(float x) { ... }  
void println(int x) { ... }  
void println(long x) { ... }  
void println(Object x) { ... }  
void println(String x) { ... }
```


■ 변수의 유효범위

변수의 종류	선언위치	생성시기
클래스 변수 (스태틱 변수)	클래스 영역	클래스가 실행될때 (main 메소드 실행될때)
인스턴스 변수 (멤버 변수)		인스턴스가 생성될때 (new 클래스() 실행될때)
지역변수	클래스 영역 외 (메소드, 생성자, for, if 등)	변수 선언문이 수행될때 (해당 영역이 실행될때)

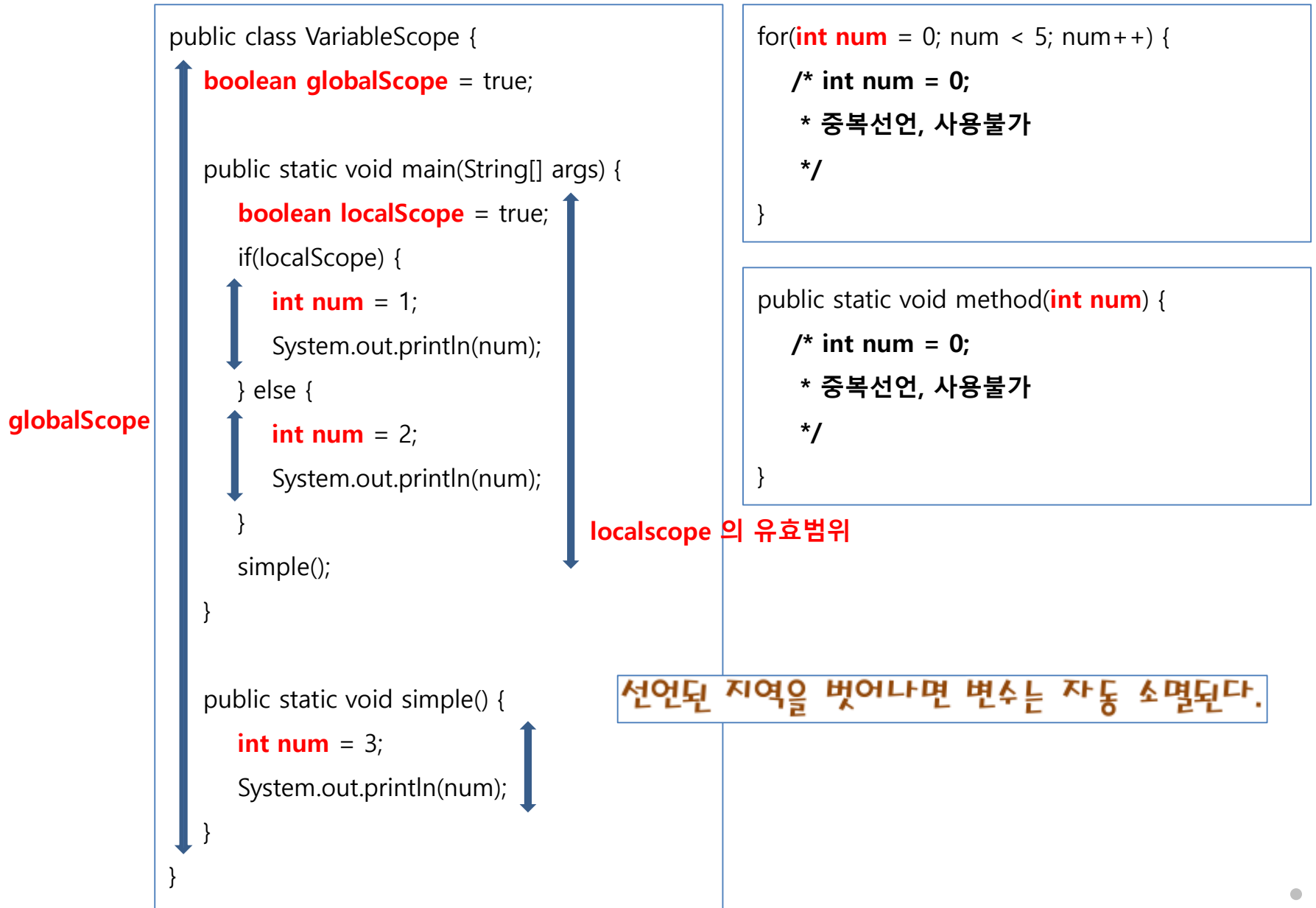
■ 변수의 유효범위

```
class Variables {  
    static int cv;        // Class Variable (Static Variable)  
    int iv;               // Instance Variable (Member Variable)  
  
    void method() {  
        int lv = 0;       // Local Variable  
    }  
}
```

클래스 영역

메소드 영역

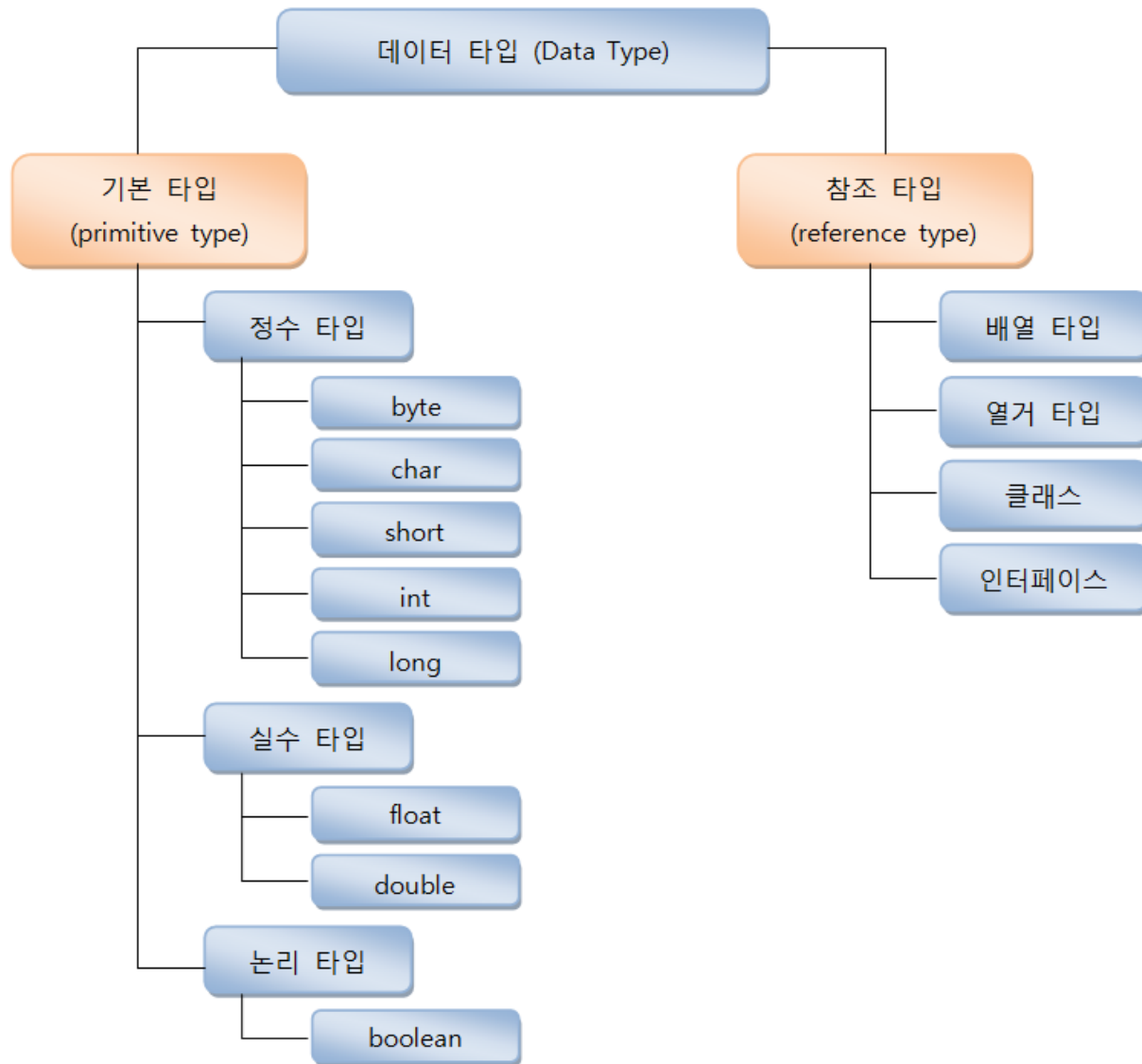
■ 변수의 유효범위



■ 변수 유효범위

```
public class MethodExam4 {  
    static int num = 10;  
  
    public static void main(String[] args) {  
        int num = 20;  
        System.out.println(num);  
    }  
  
    public static void temp() {  
        int num = 30;  
        System.out.println(num);  
    }  
}
```

■ 데이터 타입 분류



■ 변수의 메모리 사용

- 기본 타입 변수 – 실제 값을 변수 안에 저장
- 참조 타입 변수 – 주소를 통해 객체 참조

[기본 타입 변수]

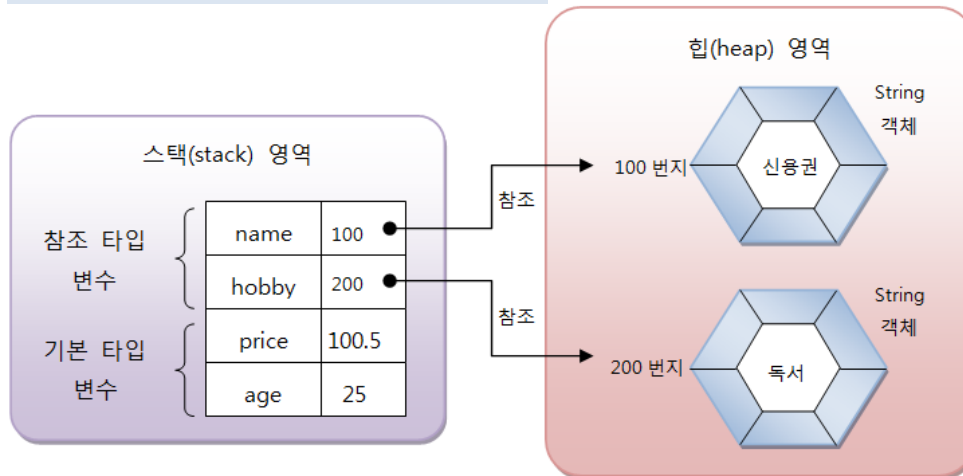
```
int age = 25;
```

```
double price = 100.5;
```

[참조 타입 변수]

```
String name = "신용권";
```

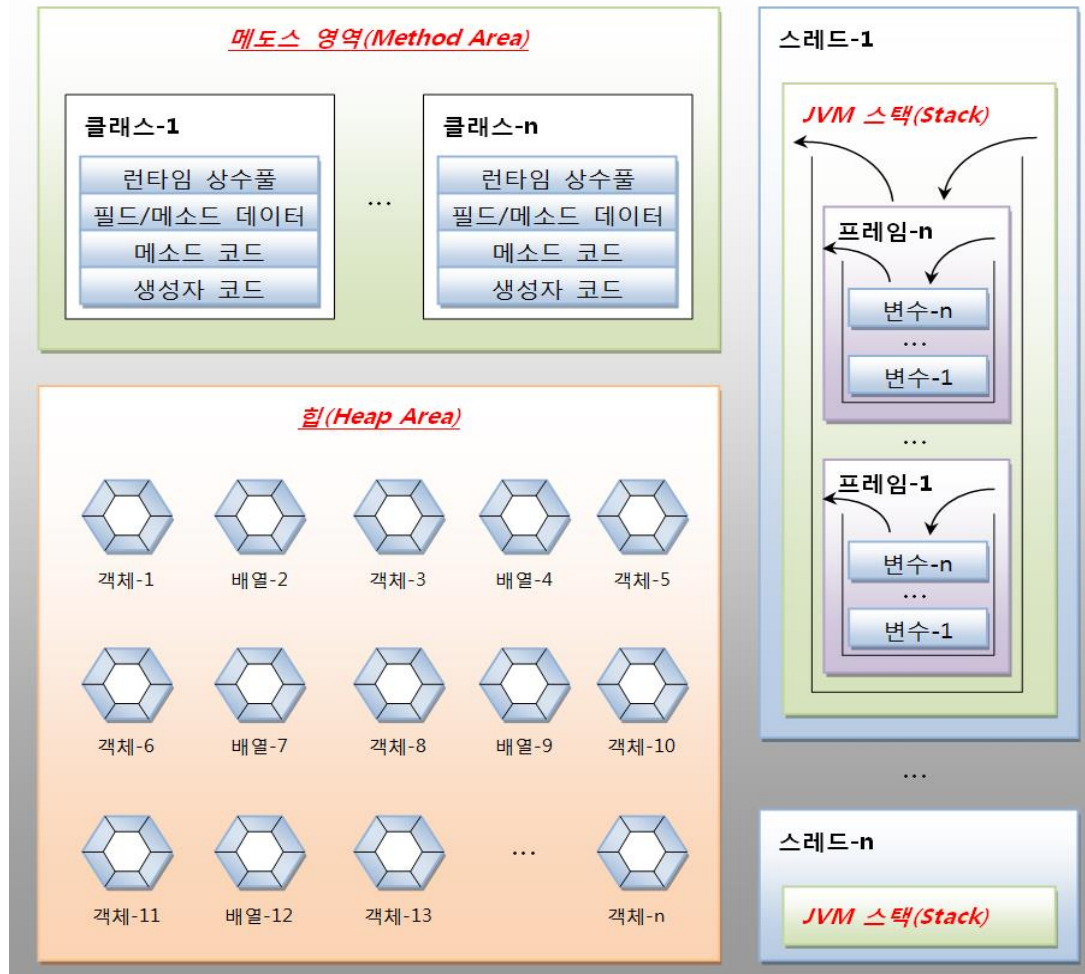
```
String hobby = "독서";
```



■ JVM이 사용하는 메모리 영역

- OS에서 할당 받은 메모리 영역(Runtime Data Area)을 세 영역으로 구분

Runtime Data Area



■ JVM이 사용하는 메모리 영역

● 메소드 영역

- JVM 시작할 때 생성
- 로딩된 클래스 바이트 코드 내용을 분석 후 저장
- 모든 스레드가 공유

● 힙 영역

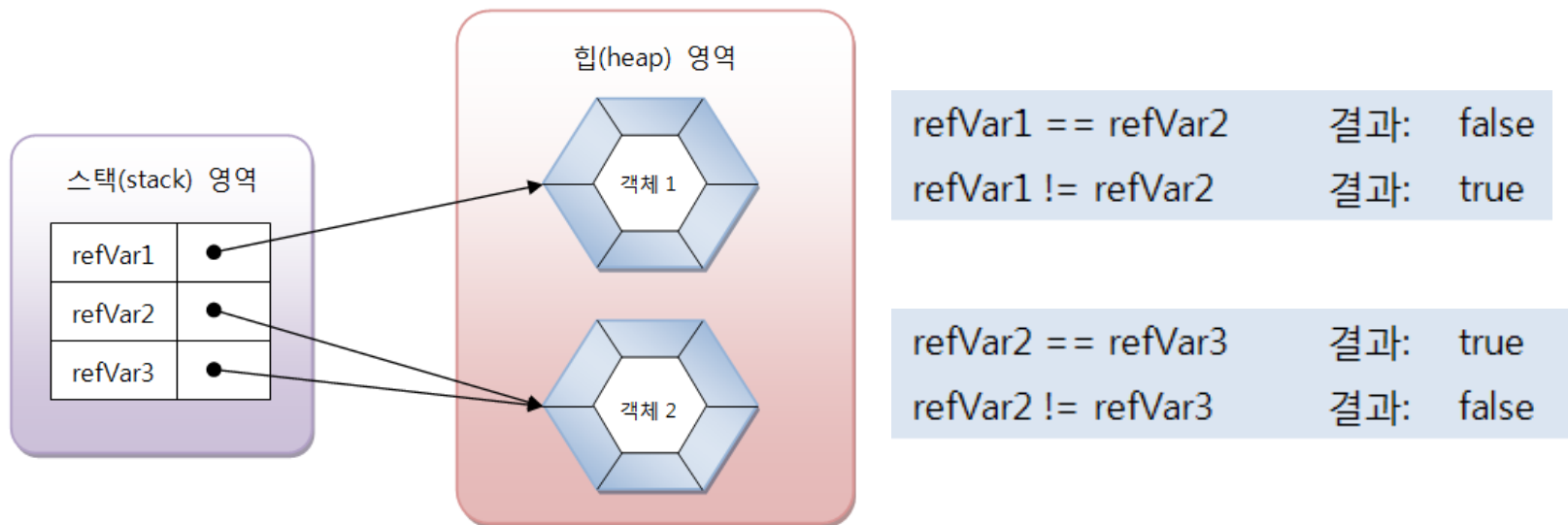
- JVM 시작할 때 생성
- 객체/배열 저장
- 사용되지 않는 객체는 Garbage Collector 가 자동 제거

● JVM 스택

- 스레드 별 생성
- 메소드 호출할 때마다 Frame을 스택에 추가(push)
- 메소드 종료하면 Frame 제거(pop)

■ 변수 비교

- 기본 타입: byte, char, short, int, long, float, double, boolean
 - 의미 : 변수의 값이 같은지 다른지 조사
- 참조 타입: 배열, 열거, 클래스, 인터페이스
 - 의미 : 동일한 객체를 참조하는지 다른 객체를 참조하는지 조사



```
if( refVar2 == refVar3 ) { ... }
```

■ 변수 비교 - 1

```
public class MethodExam5 {  
    public static void main(String[] args) {  
        int num1 = 10;  
        int num2 = 10;  
        System.out.println(num1 == num2);  
  
        int[] nums1 = {1, 2};  
        int[] nums2 = {1, 2};  
        System.out.println(nums1 == nums2);  
        System.out.println(nums1.hashCode());  
        System.out.println(nums2.hashCode());  
  
        nums1 = nums2;  
        System.out.println(nums1 == nums2);  
        System.out.println(nums1.hashCode());  
        System.out.println(nums2.hashCode());  
    }  
}
```

■ 변수 비교 – 2 (Call by Value / Call by Reference)

```
public class MethodExam6 {  
    public static void main(String[] args) {  
        int num = 10;  
  
        int[] nums = {10, 20, 30};  
  
        changeValue(num); // call by value  
        System.out.println("호출 후 : " + num);  
  
        changeValue(nums); // call by reference  
        System.out.println("호출 후 : " + nums[0]);  
    }  
}
```

```
static void changeValue(int num) {  
    num = num * 10;  
    System.out.println("1번 : " + num);  
}  
  
static void changeValue(int[] nums) {  
    nums[0] = nums[0] * 10;  
    System.out.println("2번 : " + nums[0]);  
}  
}
```