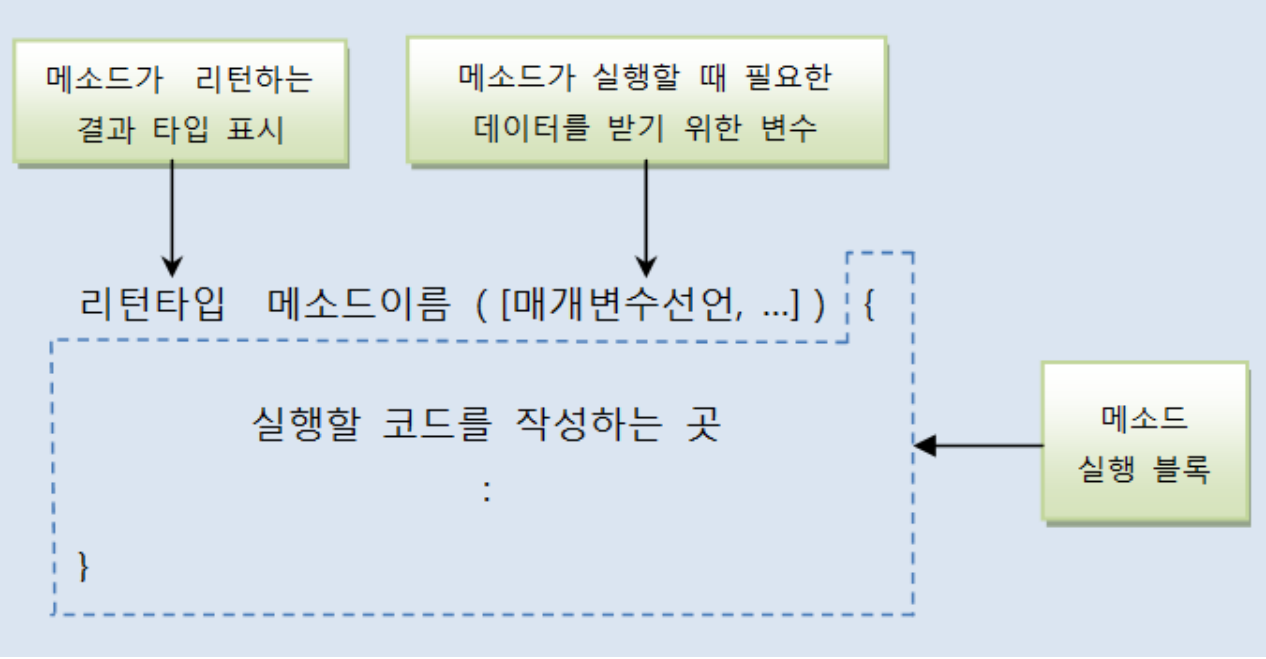
■ 메소드

- 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 */
        }
    }
}
```

■ 새로운 메소드 만들기

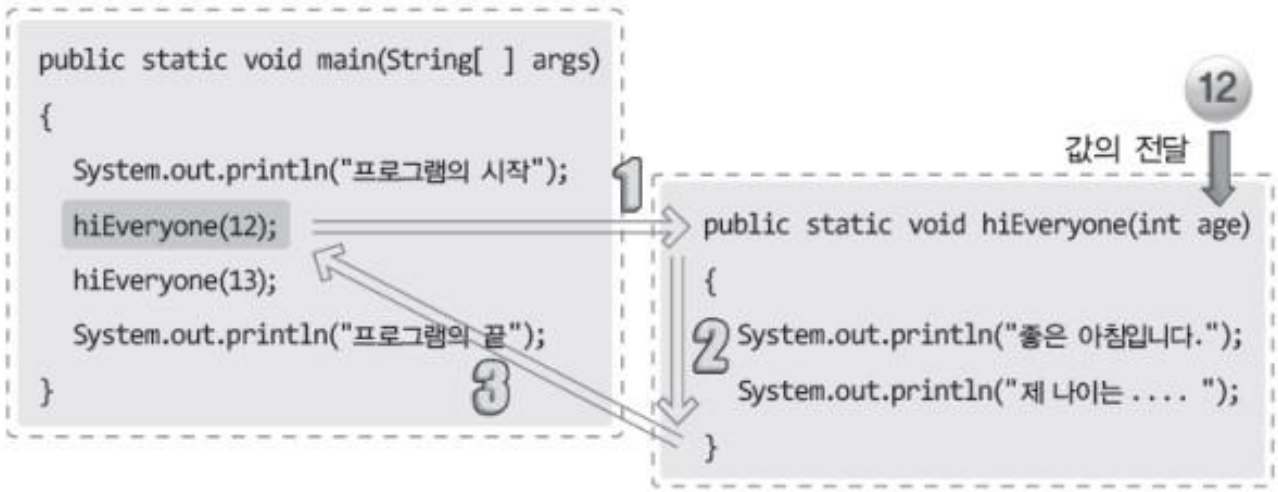
```
class MethodDefAdd
{
    public static void main(String[] args)
    {
        System.out.println("프로그램의 시작");
        hiEveryone(12);
        hiEveryone(13);
        System.out.println("프로그램의 끝");
    }
    public static void hiEveryone(int age)
    {
        System.out.println("좋은 아침입니다.");
        System.out.println("제 나이는 "+ age+"세입니다.");
    }
}
```

메소드 실행(호출) 방법

매개변수

실행결과

```
프로그램의 시작
좋은 아침입니다.
제 나이는 12세입니다.
좋은 아침입니다.
제 나이는 13세입니다.
프로그램의 끝
```



■ 매개변수가 두 개인 형태의 메소드

```
class Method2Param
{
    public static void main(String[] args)
    {
        double myHeight=175.9;
        hiEveryone(12, 12.5);
        hiEveryone(13, myHeight);
        byEveryone();
    }
    public static void hiEveryone(int age, double height)
    {
        System.out.println("제 나이는 "+ age+"세 입니다.");
        System.out.println("저의 키는 "+ height+"cm 입니다.");
    }
    public static void byEveryone()
    {
        System.out.println("다음에 뵙겠습니다.");
    }
}
```

전달 순서대로 저장

전달되는 것 없음

실행결과

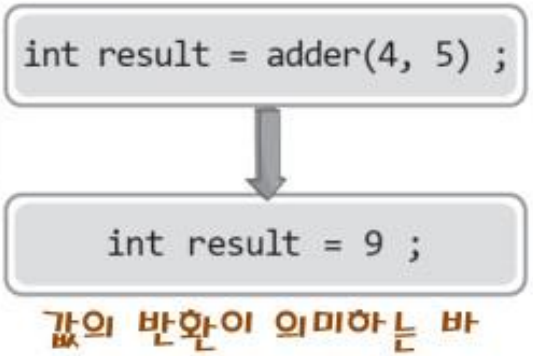
```
제 나이는 12세 입니다.
저의 키는 12.5cm 입니다.
제 나이는 13세 입니다.
저의 키는 175.9cm 입니다.
다음에 뵙겠습니다.
```

■ 값을 반환하는 메소드

```
class MethodReturns
{                                  값을 반환하지 않겠다.
    public static void main(String[] args)
    {
        int result=adder(4, 5);
        System.out.println("4와 5의 합 : " + result);
        System.out.println("3.5의 제곱 : " + square(3.5));
    }                          int형 데이터를 반환하겠다.
    public static int adder(int num1, int num2)
    {
        int addResult=num1+num2;
        return addResult;
    }                          double형 데이터를 반환하겠다.
    public static double square(double num)
    {
        return num*num;
    }
}
```

실행결과

```
4와 5의 합 : 9
3.5의 제곱 : 12.25
```



값의 반환이 의미하는 바

■ 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;
    }
}
```

■ 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 실행을 강제 종료 시키는 역할

■ 반환값이 없는 메소드

```
class OnlyExitReturn
{
    public static void main(String[] args)
    {
        divide(4, 2);
        divide(6, 2);
        divide(9, 0);
    }
    public static void divide(int num1, int num2)
    {
        if(num2==0)
        {
            System.out.println("0으로는 값을 나눌 수 없습니다.");
            return;
        }
        System.out.println("나눗셈 결과 : " + (num1/num2));
    }
}
```

메소드의 종료만을 의미함

실행 결과

```
나눗셈 결과 : 2
나눗셈 결과 : 3
0으로는 값을 나눌 수 없습니다.
```

값의 반환, 메소드의 종료, 이렇게 두 가지의 의미를 지님

■ 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

globalScope

```
public class VariableScope {
    boolean globalScope = true;

    public static void main(String[] args) {
        boolean localScope = true;
        if(localScope) {
            int num = 1;
            System.out.println(num);
        } else {
            int num = 2;
            System.out.println(num);
        }
        simple();
    }

    public static void simple() {
        int num = 3;
        System.out.println(num);
    }
}
```

localscope 의 유효범위

```
for(int num = 0; num < 5; num++) {
    /* int num = 0;
     * 중복선언, 사용불가
     */
}
```

```
public static void method(int num) {
    /* int num = 0;
     * 중복선언, 사용불가
     */
}
```

선언된 지역을 벗어나면 변수는 자동 소멸된다.

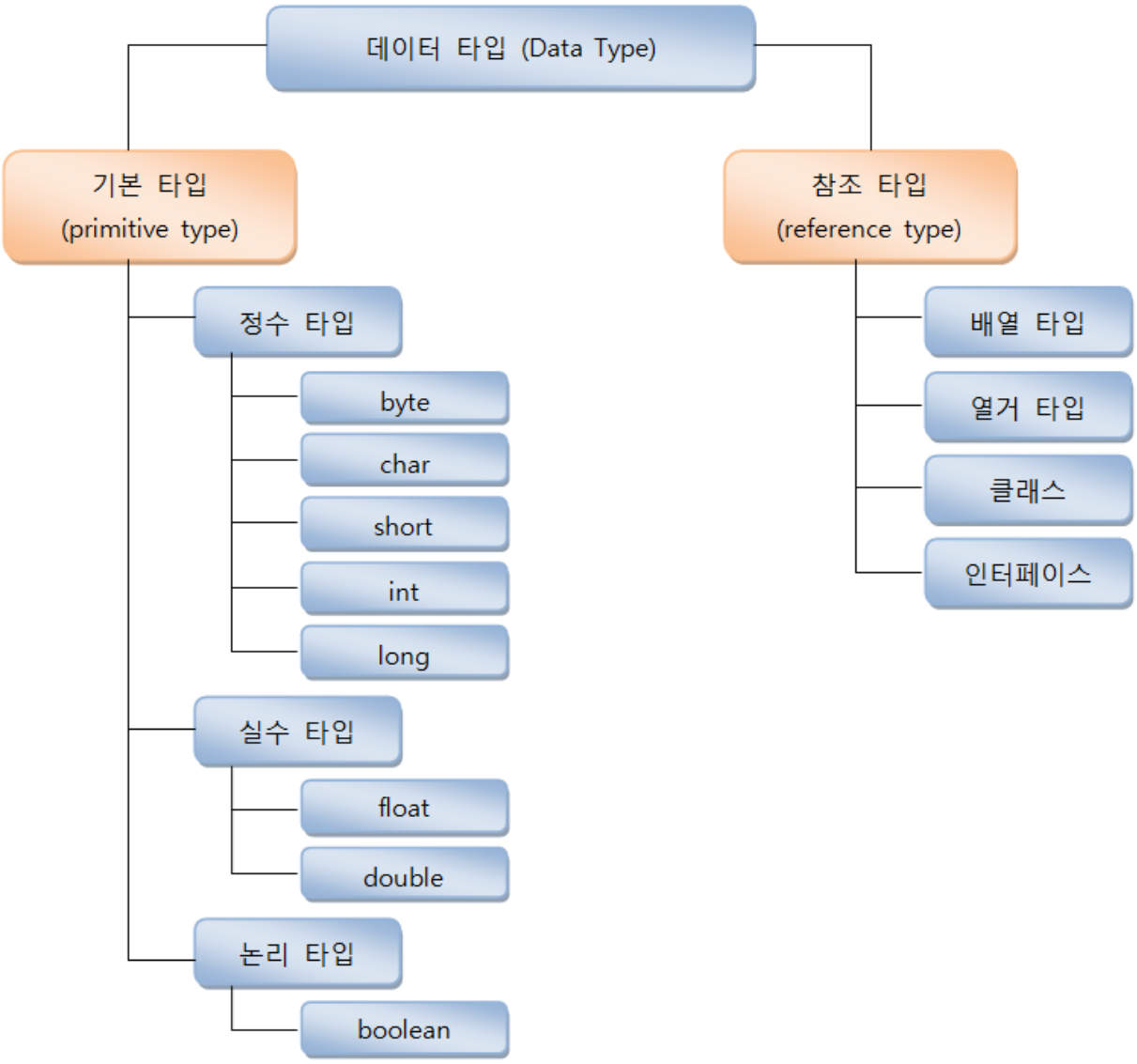
■ 변수 유효범위

```
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

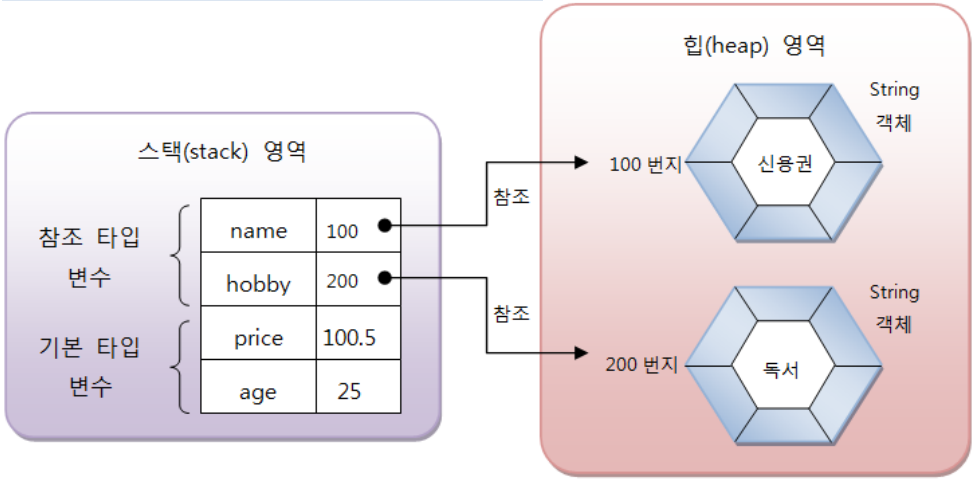

데이터 타입 (Data Type)
기본 타입
(primitive type)
참조 타입
(reference type)
정수 타입
byte
char
short
int
long
실수 타입
float
double
논리 타입
boolean
배열 타입
열거 타입
클래스
인터페이스

■ 변수의 메모리 사용

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

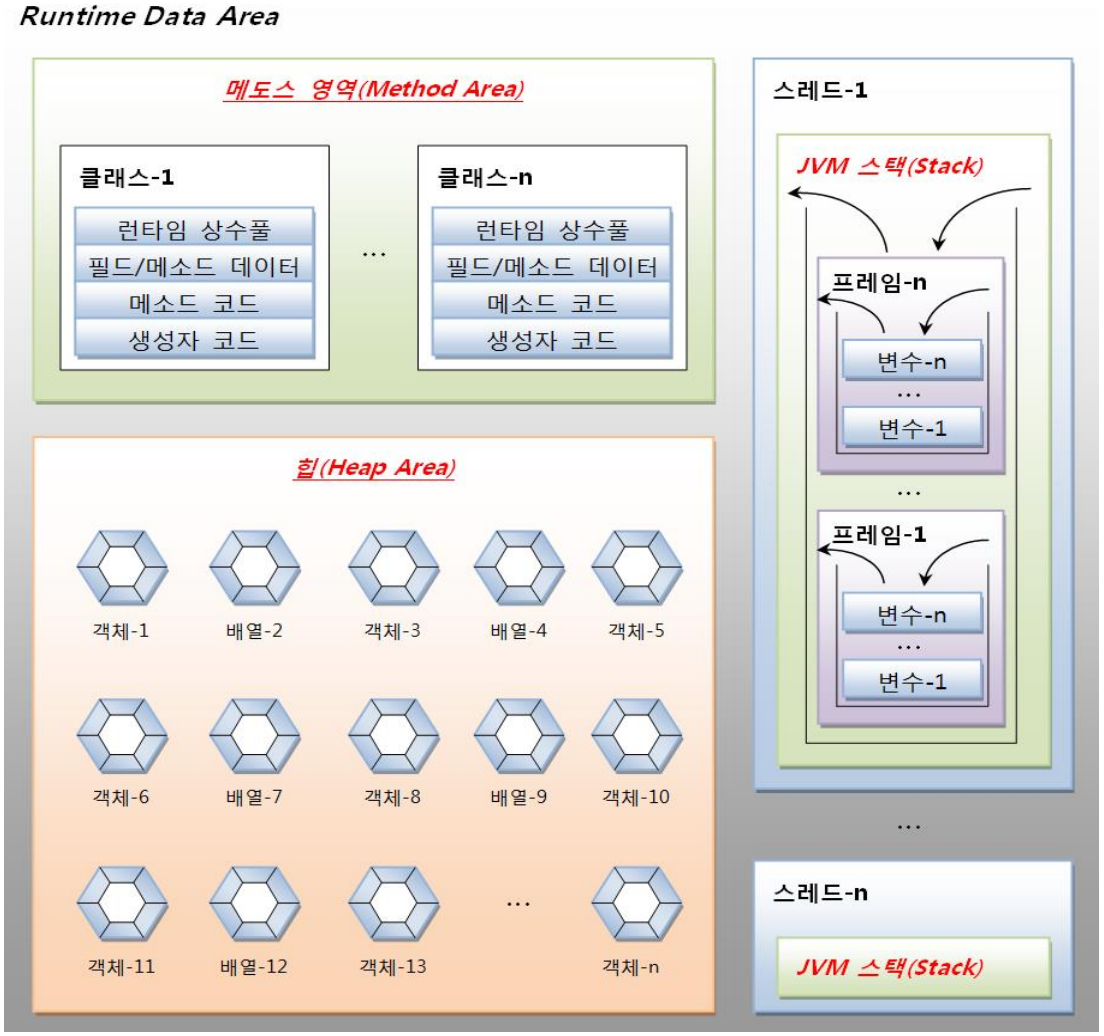
```
[기본 타입 변수]
int age = 25;
double price = 100.5;

[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



■ JVM이 사용하는 메모리 영역

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

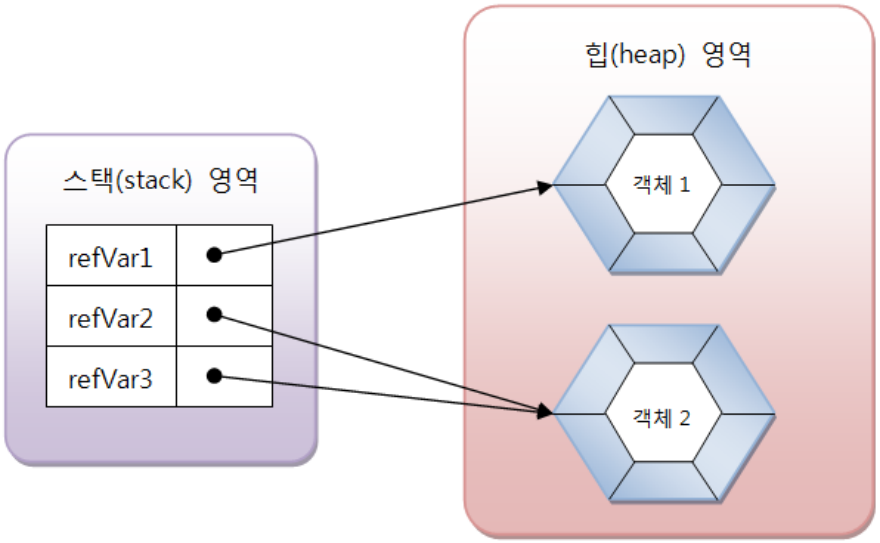
● 힙 영역
- JVM 시작할 때 생성
- 객체/배열 저장
- 사용되지 않는 객체는 Garbage Collector 가 자동 제거

● 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




```
refVar1 == refVar2      결과:  false
refVar1 != refVar2      결과:  true
```

```
refVar2 == refVar3      결과:  true
refVar2 != refVar3      결과:  false
```

```
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]);
    }
}
```