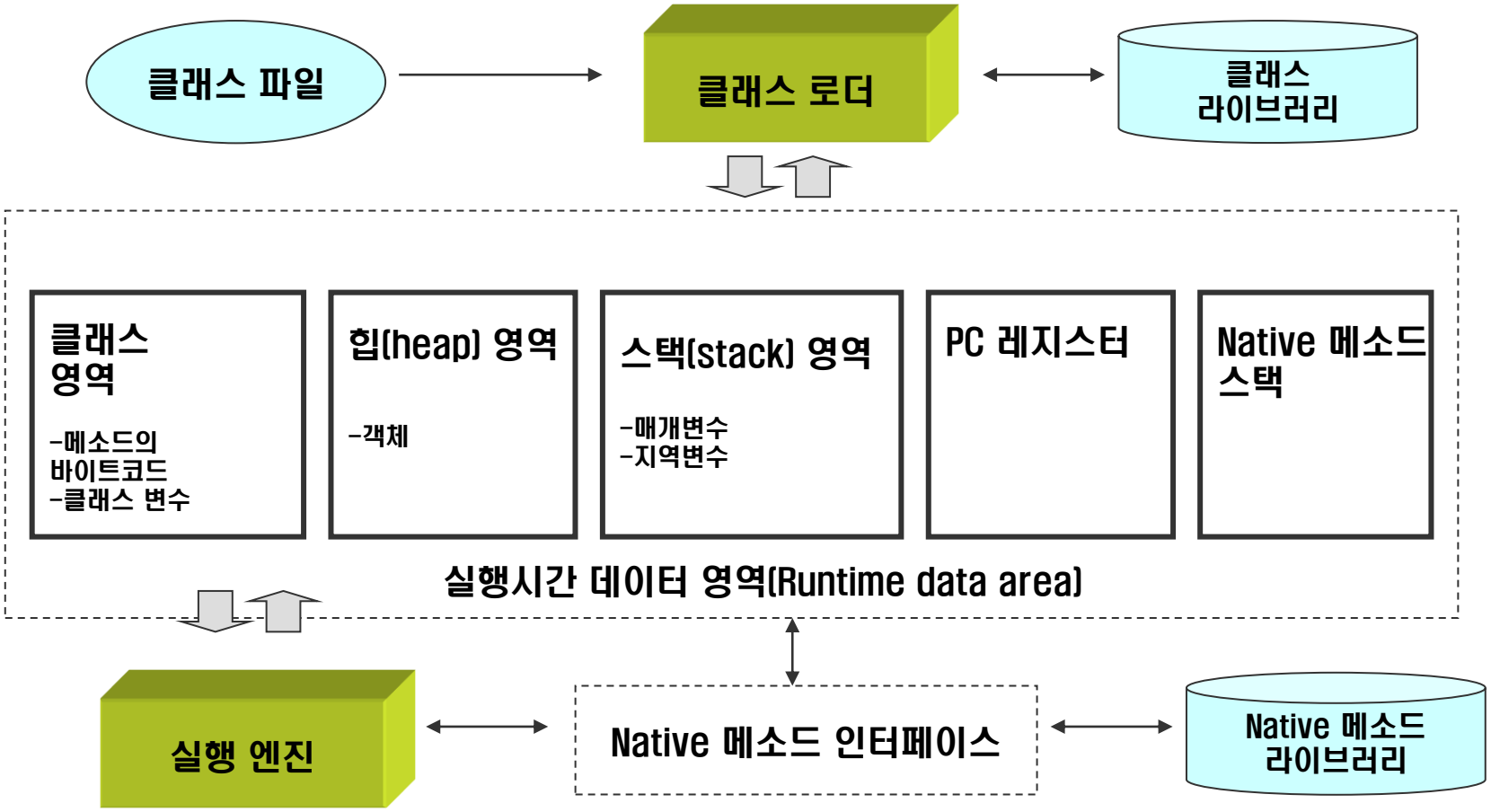
JVM과 메모리

JVM의 구조와 메모리 모델
Exam
(1) Static, stack
(2) Heap의 Instance
(3) Array

JVM의 구조와 메모리 모델

- JVM의 내부 구조



JVM의 구조와 메모리 모델

1) 클래스 영역 == 메소드 영역 == 스태틱 영역
   - 메소드와 클래스 변수를 저장하기 위한 공간
   - 모든 프로그램에 의해 공유
   - 이 영역도 자료구조는 Heap으로 되어있음.
2) 힙 영역(Heap area)
   - 동적으로 할당하여 사용할 수 있는 메모리
   - 주로 실행시간에 생성되는 객체를 저장
3) 스택 영역(Stack area)
   - 메소드 호출 시 메소드의 매개변수, 지역변수, 임시변수등을 저장하기 위한 스택 구조의 메모리
   - 실행중인 프로그램에 따라 스택 프레임 할당
4) PC 레지스터
   - JVM이 현재 수행할 명령어의 주소를 저장
5) Native 메소드 스택
   - Native 메소드를 호출할 때 native 메소드의 매개변수, 지역변수 등을 저장

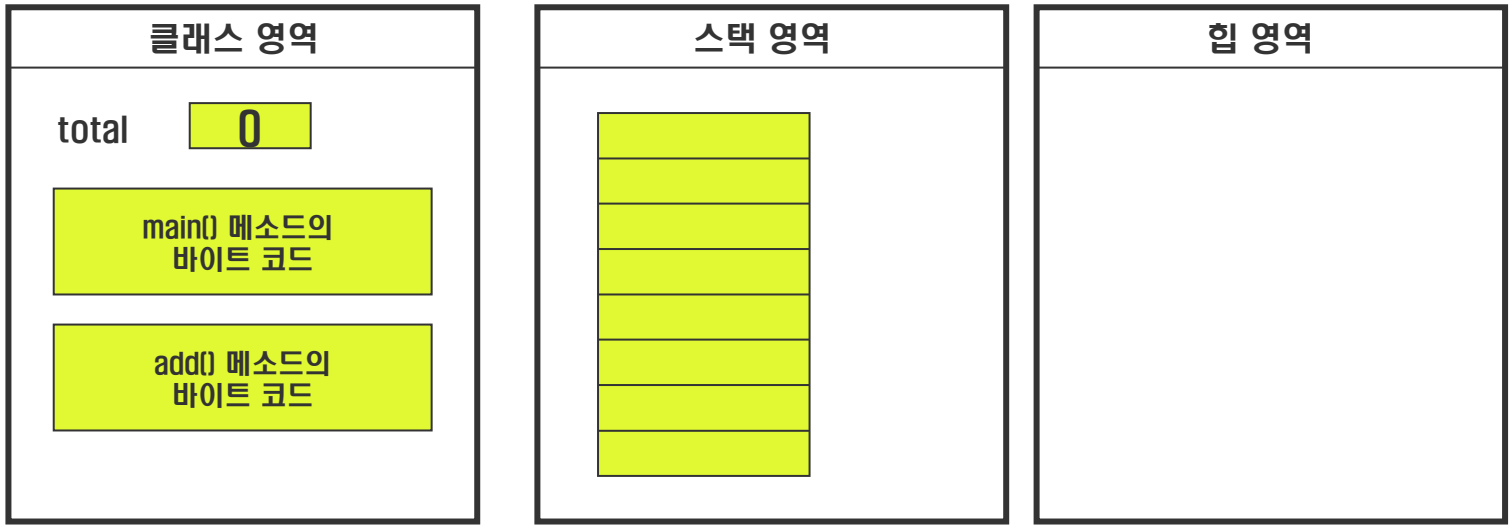
Exam.1 static, stack

```
class MemoryTest {
    static int total = 0;
    public static void main(String[] args) {
        int x=10, y=20, z;  //1번
        z=add(x,y);
    }
    static int add(int a, int b) {
        total++;     //2번
        return(a+b);
    }
}
```

MemoryTest.java

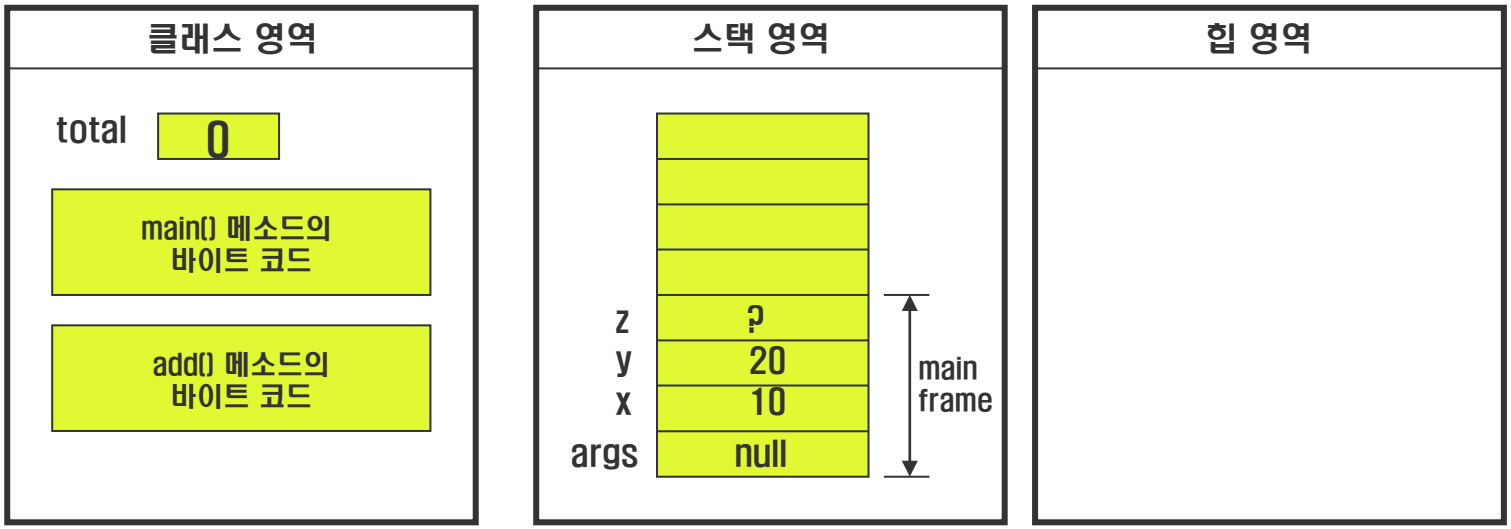
Static, stack
(1) 클래스 파일의 로딩 시점

- 두 개의 메소드와 클래스 변수가 클래스 영역에 저장



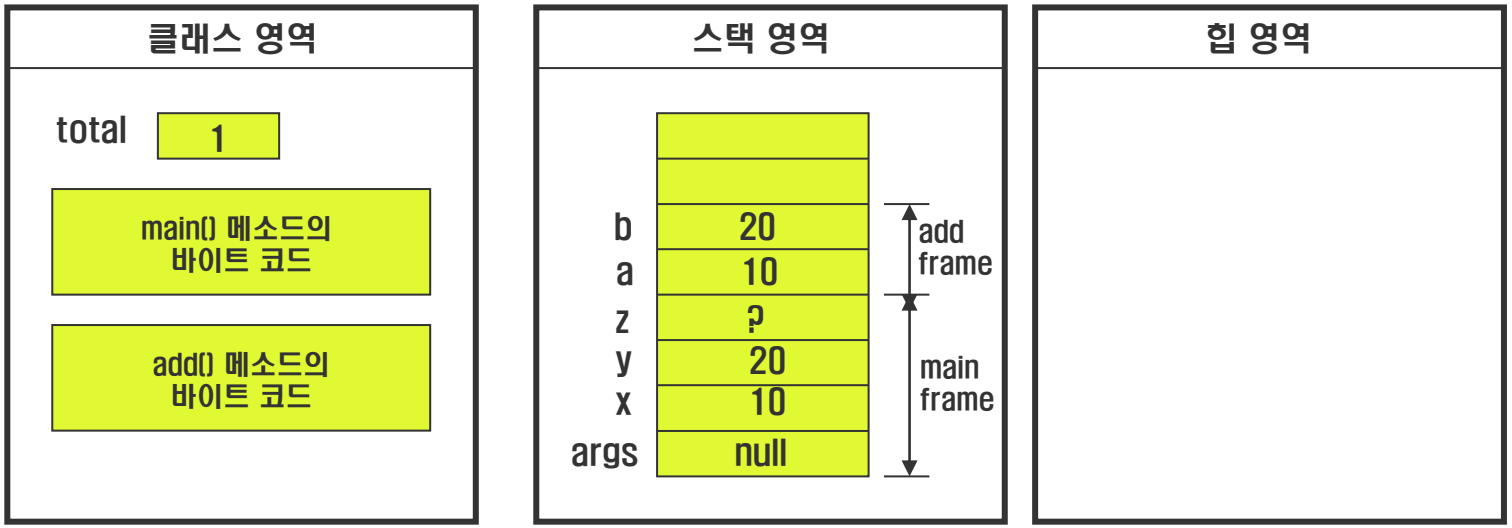
static, stack
(2) main() 메소드의 호출 시점

- “1번”위치
- main() 메소드가 호출되고 지역변수 x,y,z가 초기화 된 상태
- args(문자열의 배열)에는 값이 주어지지 않았으므로 null



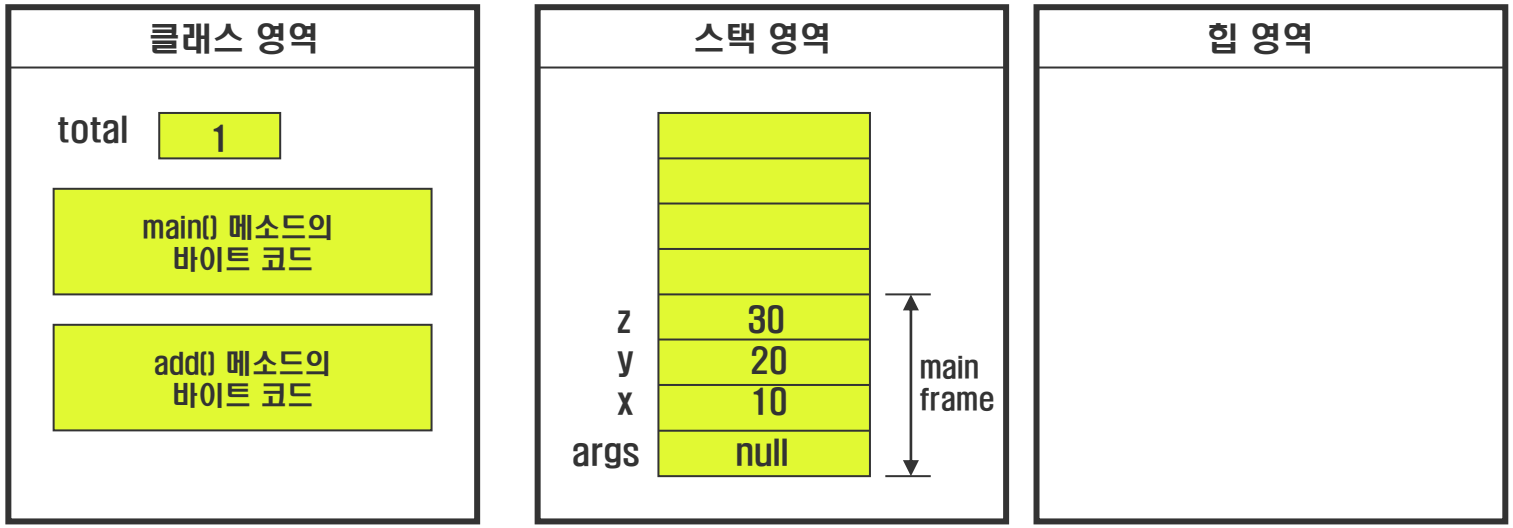
static, stack
(3) add() 메소드의 수행시점

- “2번”위치
- add() 메소드를 위한 스택 프레임 할당
- add() 메소드의 지역 변수 할당
- total 값 증가



Static, stack
(4) add() 메소드의 반환 시점

- add() 메소드의 수행 결과가 z에 저장되고 add() 메소드를 위한 프레임 제거



Exam.2 Heap의 Instance

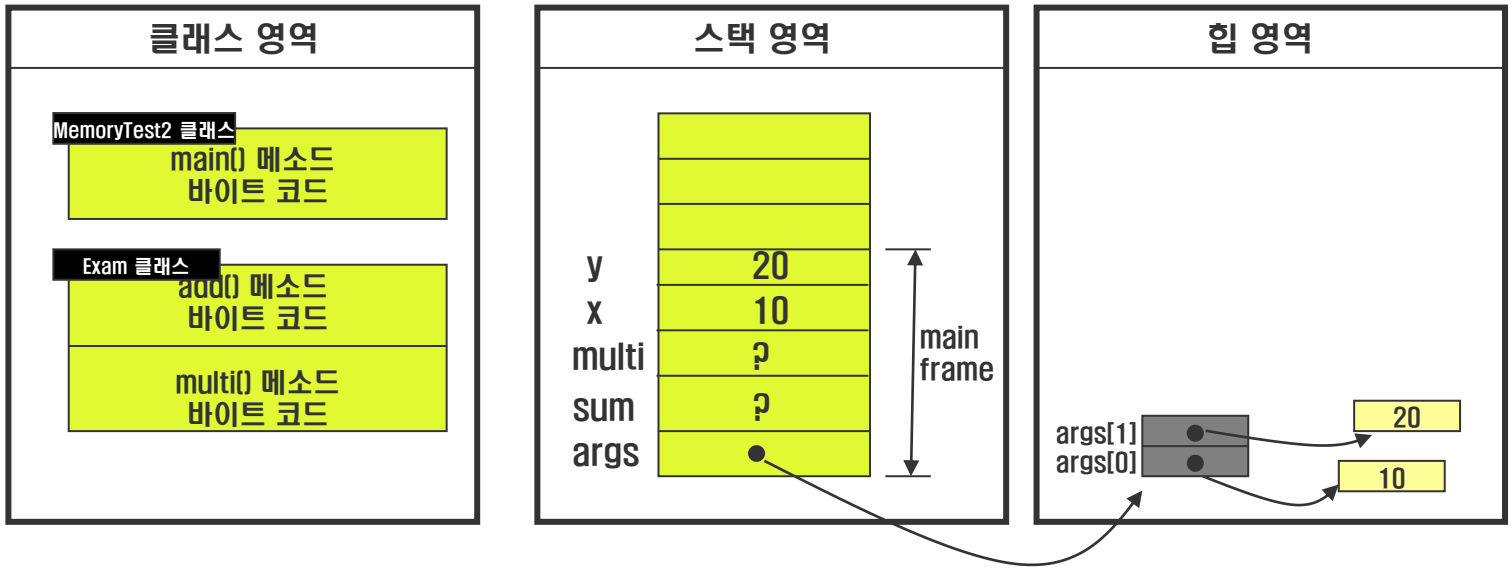
```
class Exam {
  int c,d;
  public int add(int a, int b) {
    c = a + b;
    return c;
  }
  public int multi(int a, int b) {
    d = a * b;
    return d;
  }
}
```

```
class MemoryTest2 {

  public static void main(String args[]) {
    int sum,multi;
    int x,y;
    x = Integer.parseInt(args[0]);
    y = Integer.parseInt(args[1]);     //1번
    Exam ob1 = new Exam();
              Exam ob2 = new Exam();   //2번
    sum = ob1.add(x,y);
    multi = ob2.multi(x,y);
    System.out.println("입력한 값의 합은" + sum + "입니다");
    System.out.println("입력한 값의 곱은" + multi + "입니다");
  }
}
```

MemoryTest2.java

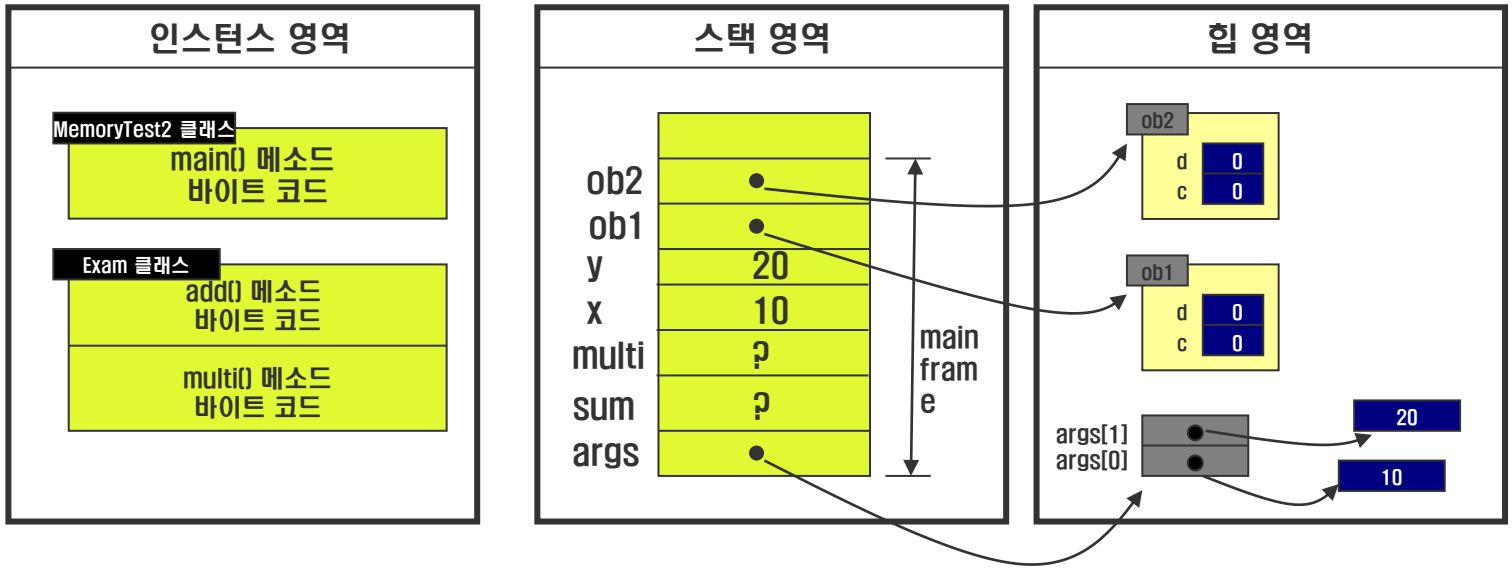
Heap의 Instance
(1) main() 메소드가 수행되는 시점

- “1번” 위치
- 프로그램 실행 시 두 개의 매개변수 지정(10,20)
- 두 개의 매개변수를 힙 영역에 문자열 객체로 저장
- x,y 에는 정수로 변환된 값이 저장



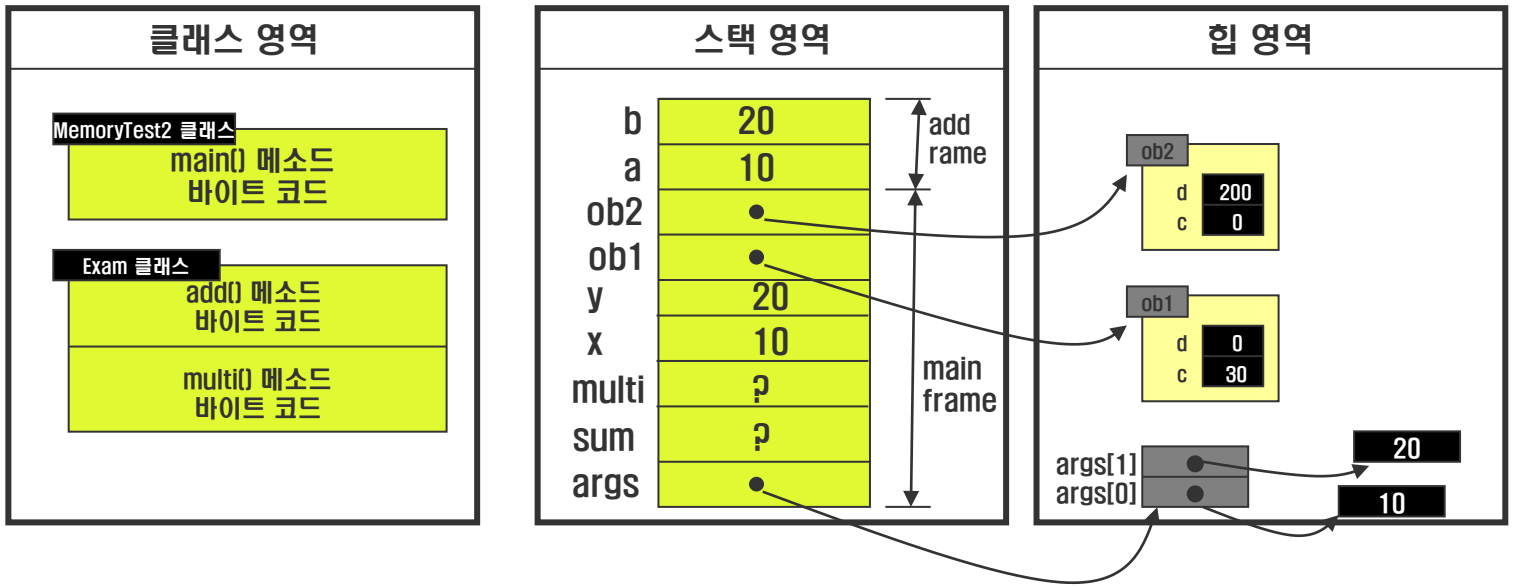
Heap의 Instance
(2) 인스턴스가 생성되는 시점

- “2번” 위치
- ob1, ob2 객체가 힙 영역에 생성
- 각각의 객체에는 두 개의 객체 속성 변수가 저장



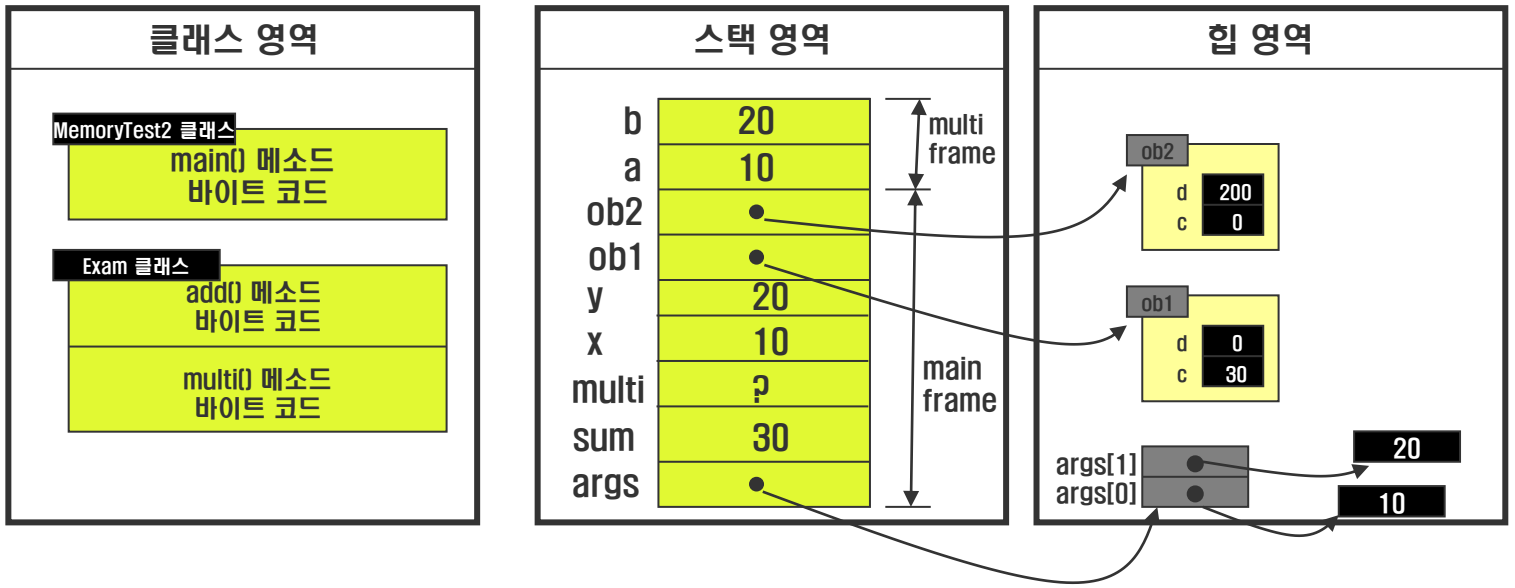
Heap의 Instance
(3) 인스턴스의 add() 메소드 수행 시점

- 두 개의 인스턴스가 생성된 다음 add() 메소드가 수행된다



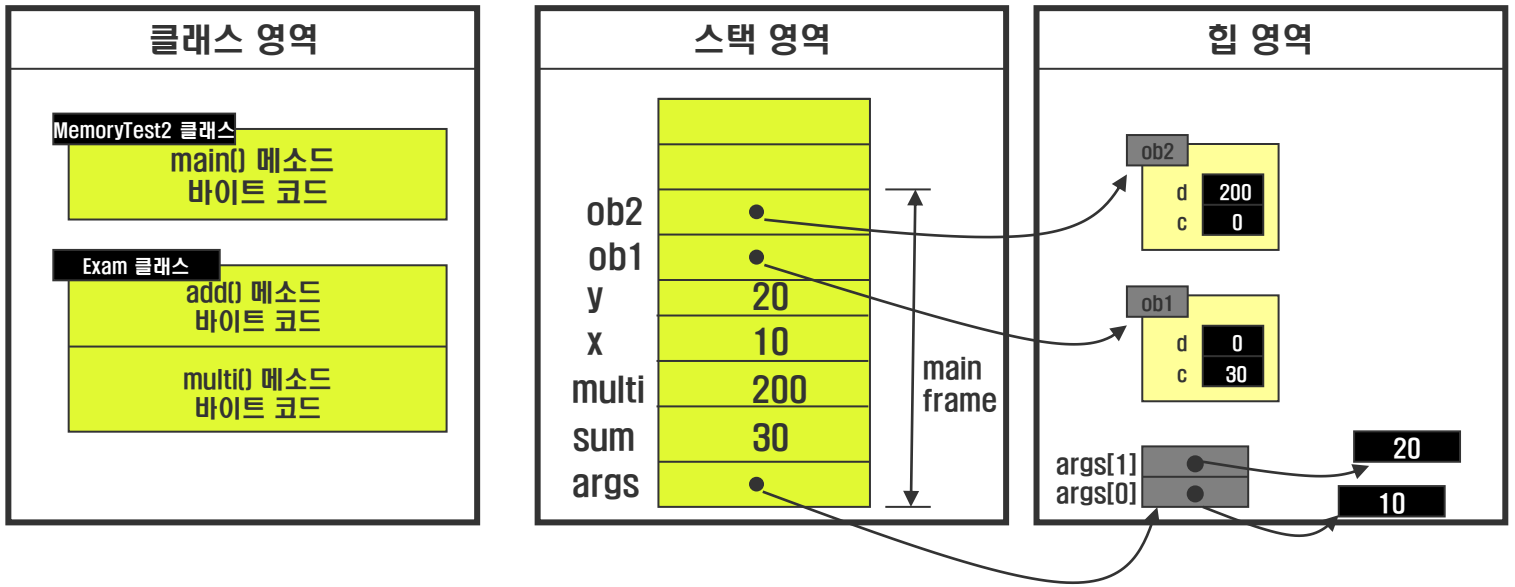
Heap의 Instance
(4) 인스턴스의 multi() 메소드 수행 시점

- add() 메소드 종료되고 30이 리턴되어 sum에 저장된다.
- Add 프레임이 제거된다.
- Multi() 메소드가 수행되며 multi frame이 생성된다.



Heap의 Instance
(5) 인스턴스의 multi() 메소드 수행 후

- multi() 메소드가 수행이 종료되며 multi frame이 제거된다.
- 200이 리턴되어 multi에 저장된다.



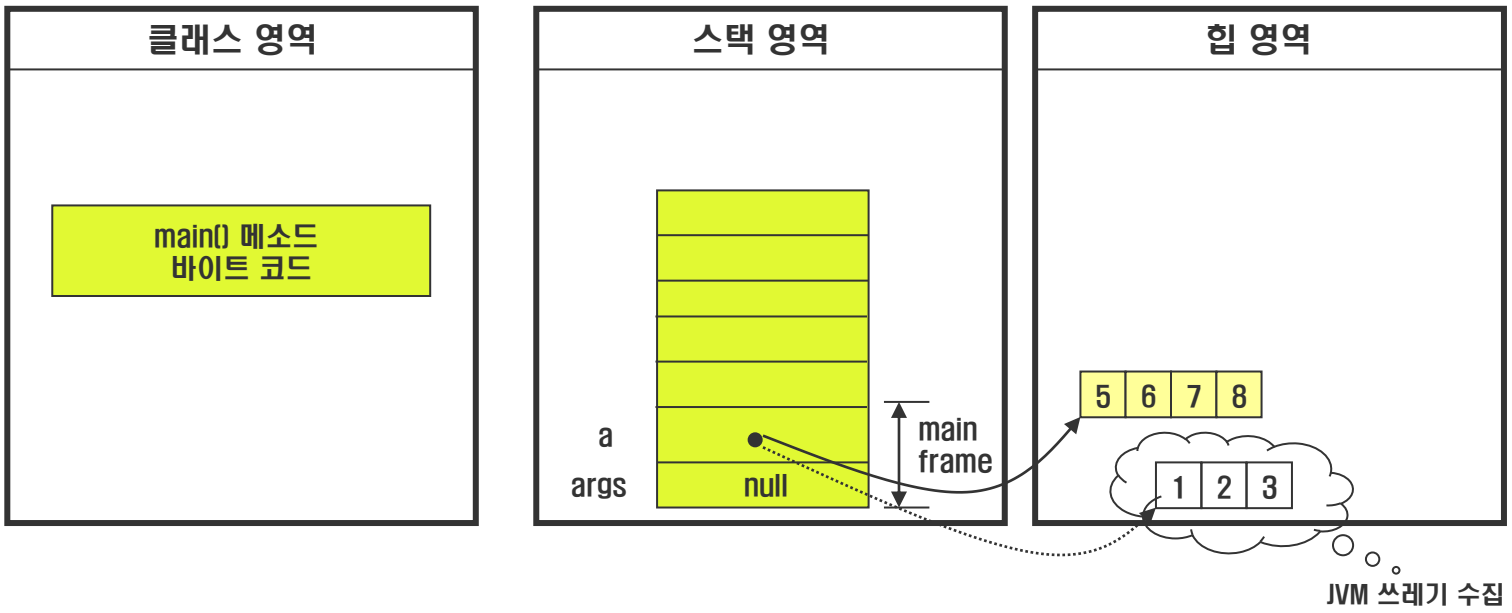
Exam 3. Array

```
class MemoryTest3 {
        public static void main(String[] args)
        {
            int a[] = new int[] {1,2,3};
            a = new int[] {5,6,7,8};
            System.out.println(a[3]);   //1번

            int b[][] = new int[][] {{1,2,3},{4,5},{6,7,8,9}};
            System.out.println(b[2][3]);   //2번

            String s1 = "Hi Java";
            System.out.println(s1);
            String s2[] = new String[] {"Hi","Java"};
            System.out.println(s2[1]);    //3번
        }
}
```

MemoryTest3.java

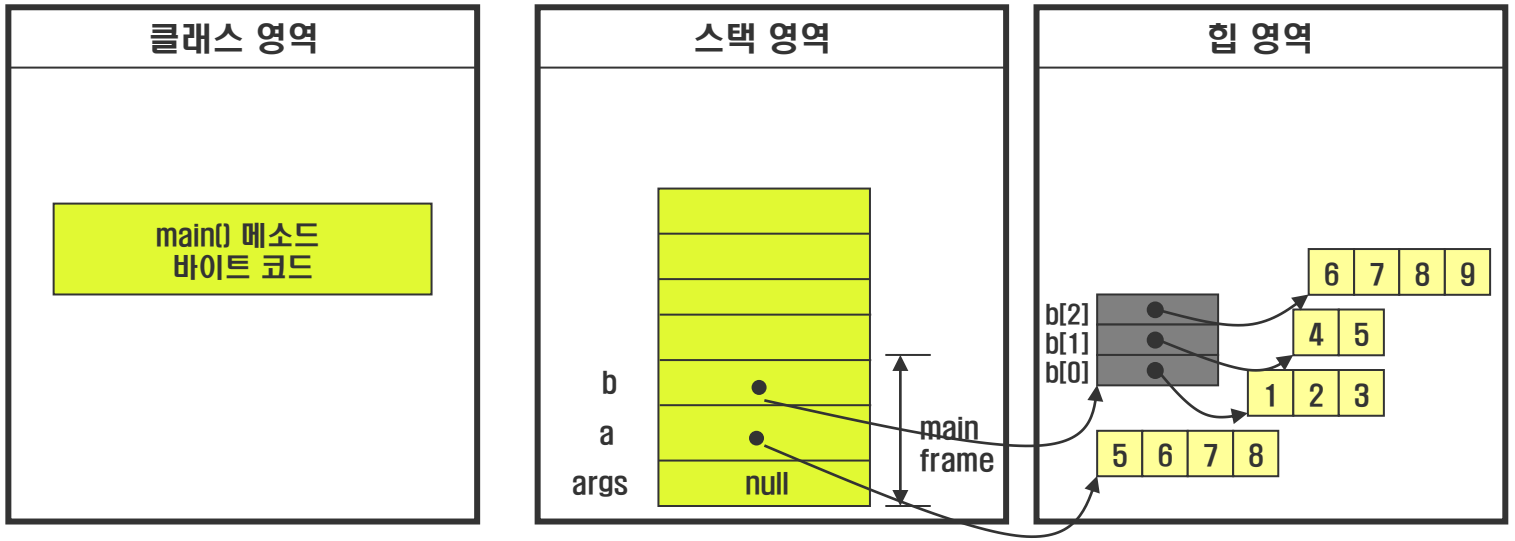
Array
(1) 배열의 생성과 변환

- “1번” 위치
- 3개의 요소를 가진 배열 a를 생성한 후 다시 배열 a에 4개의 요소를 가진 배열을 할당
- 처음 생성된 3개의 요소를 가진 배열은 쓰레기 수집된다



Array
(2) 2차원 배열과 문자열의 생성

- "2번" 위치
- 다차원 배열은 1차원 배열의 배열로 나타낸다



Array
(3) 문자열과 문자열 배열의 생성

- “3번” 위치
- 문자열을 인스턴스로 취급하여 힙 영역에 저장
- 문자열의 배열은 문자열 인스턴스의 배열로 취급