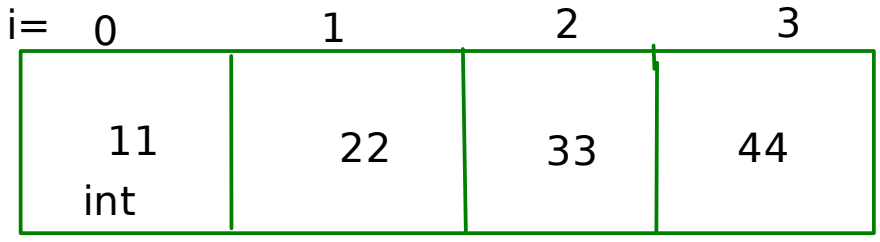
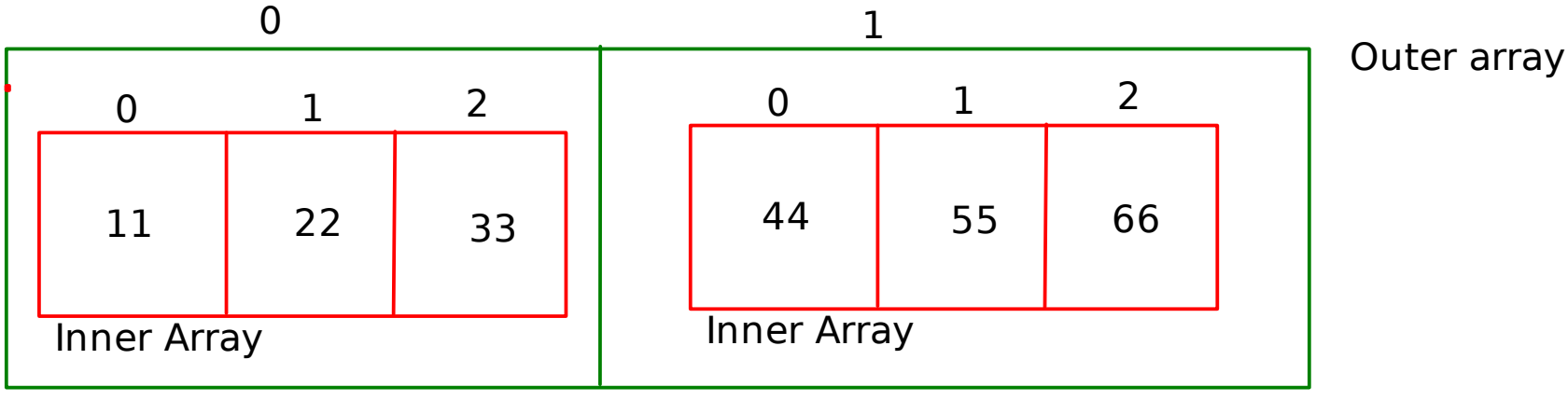
Array
1. Single Dimension
2. Multi Dimension
3. Ragged

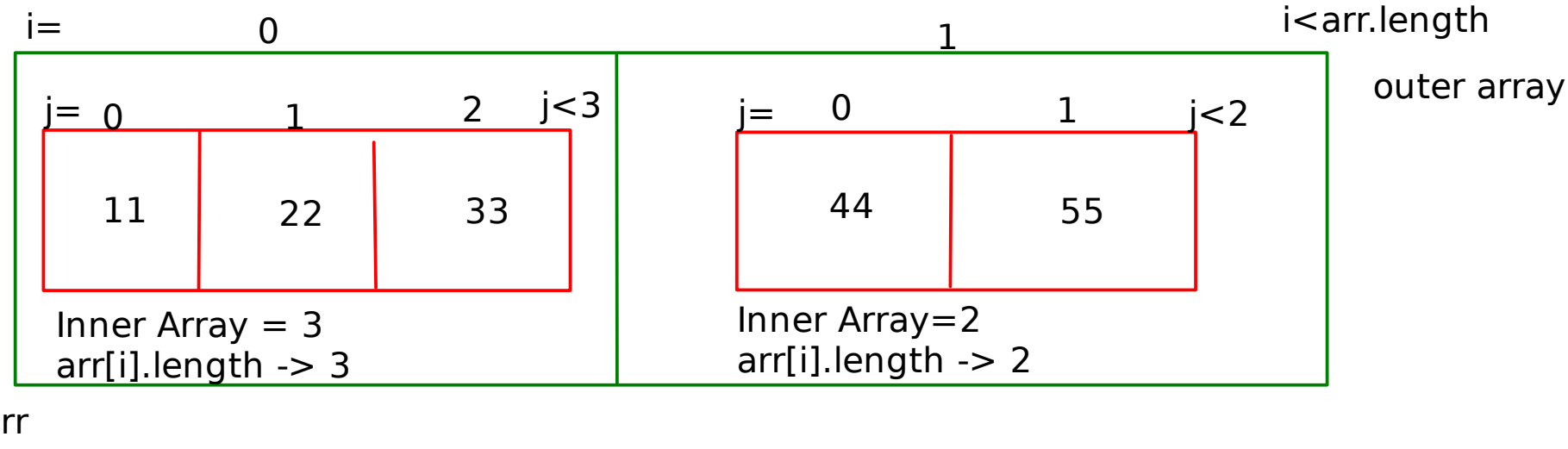



```
int []arr; // reference
arr = new int[5];
```

```
int [][] arr = new int[2][3];
```




```
int [][] arr = new int[2][];
arr[0] = new int[3];
arr[1] = new int[2];
```




```
for(int i=0;i<arr.length;i++)
        int element = arr[i];
        sysout(element);
```

```
for(int element : arr)
        sysout(element)
```

```
for(int[] ele: arr)
sysout(Arrays.toString(ele))
```

```
for(int i =0 i<arr.length;i++) // outer array
        for(int j=0 j<arr[i].length;j++)
                int element = arr[i][j];
                sysout(element)
```

```
for(int[] ele : arr)
        for(int element: ele)
                sysout(element)
```

Variable Arity Method

```
void add(int ...arr){
int result = 0;
for(int element : arr)
        result= result + element;
sysout(result);
}
```

```
add(10,20);
add(10,20,30);
```

```
int []arr = {10,20}
add(arr);

int []arr1 = {10,20,30}
add(arr1);

int []arr2 = {10,20,30,40}
add(arr2);

int []arr3 = {10,20,30,40,50}
add(arr3);
```

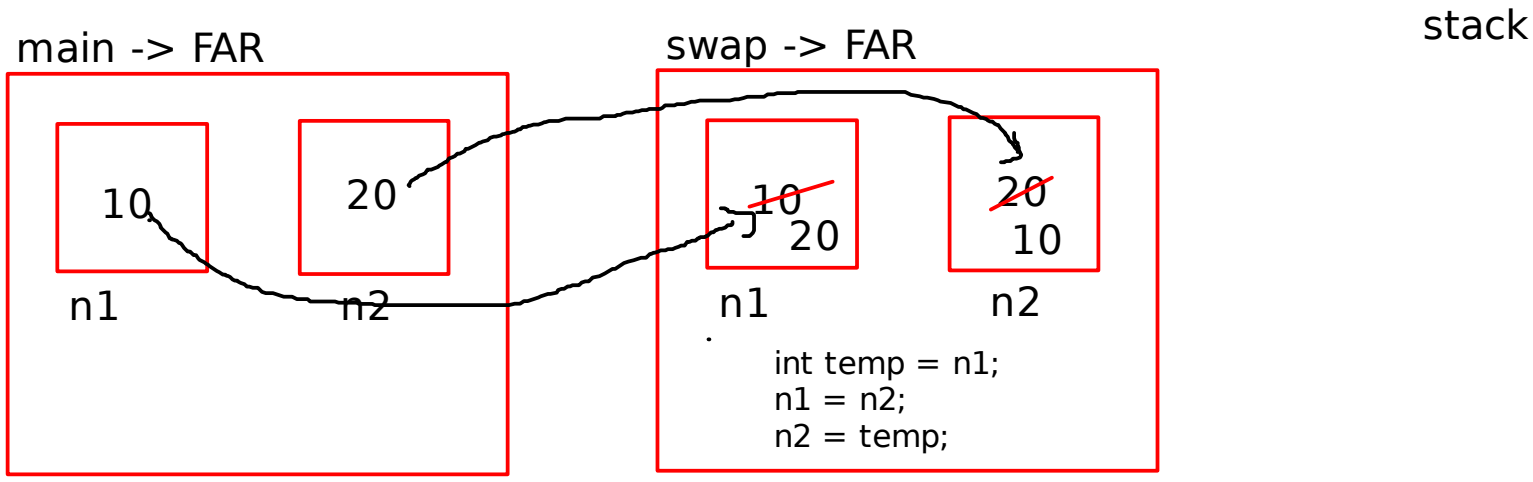
```
add(10,20);
add(10,20,30);
add(10,20,30,40);
add(10,20,30,40,50);
```

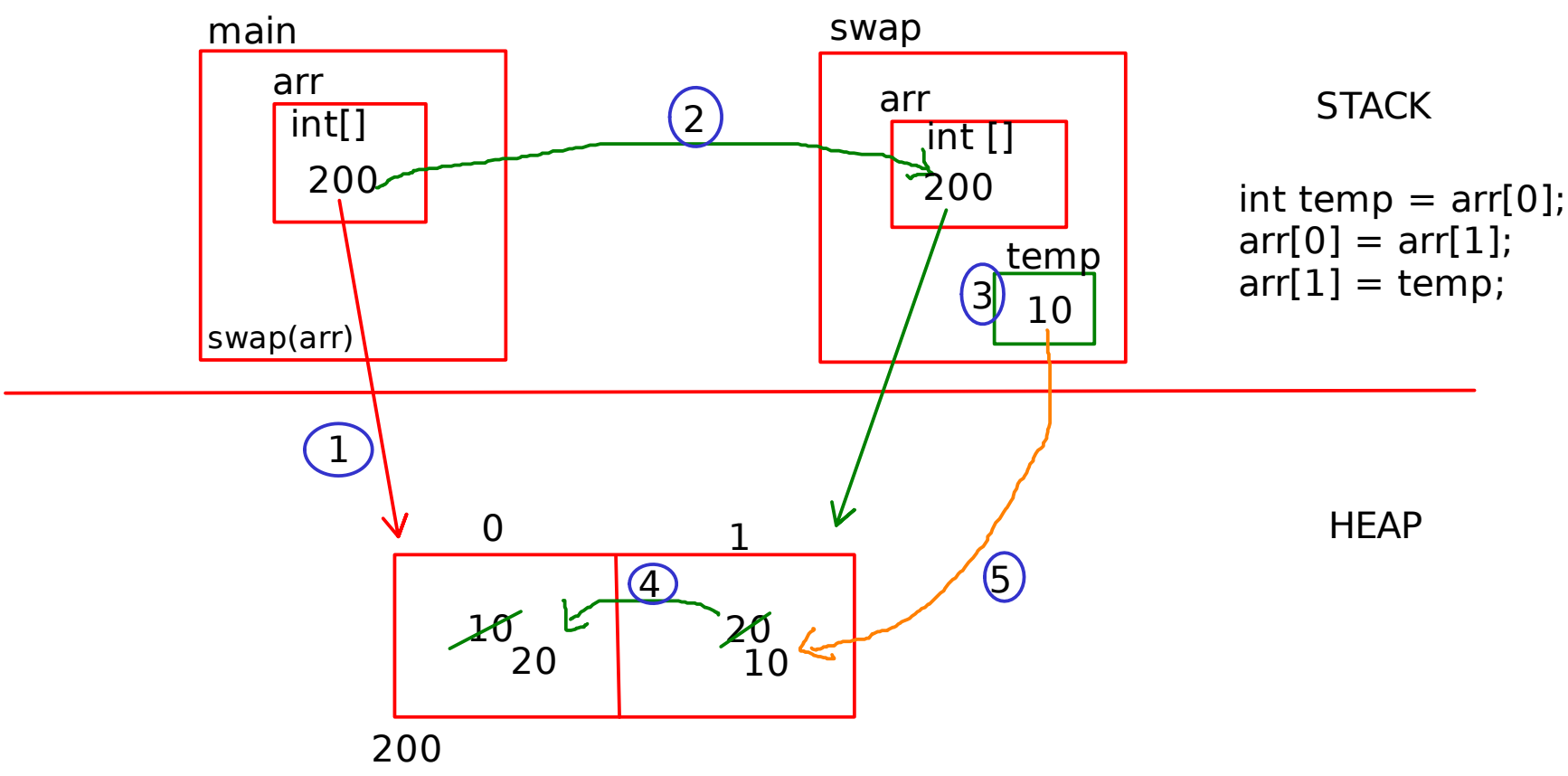
int arr[] = {11,55,33,44,22}

String getDeatils();

boolean findEmployee()

Method Arguments

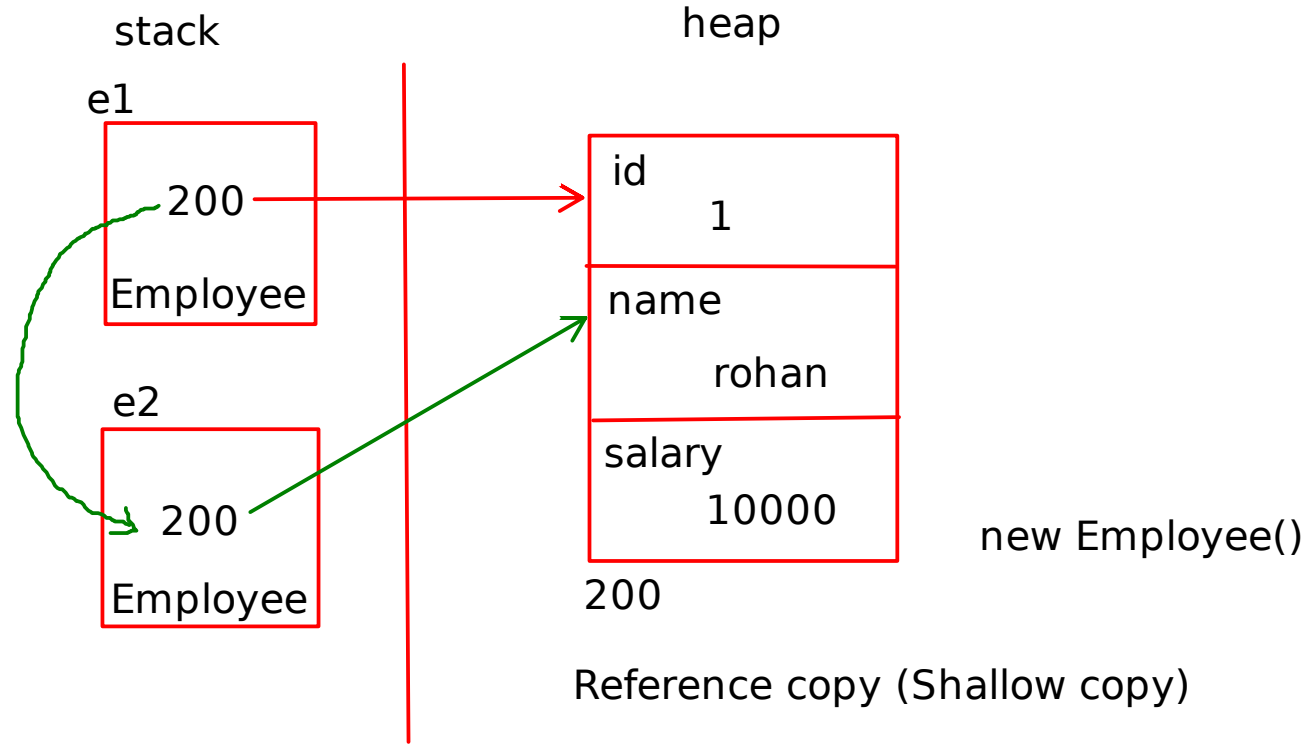




class Employee{
id
name
salary
}

Employee e1; // reference
e1 = new Employee();

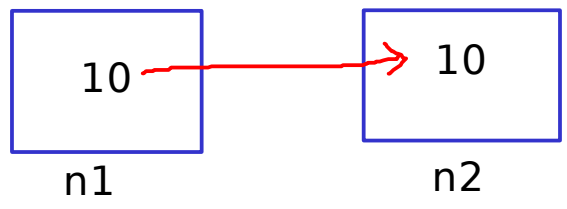
Employee e2;//reference
e2 = e1;



Reference copy (Shallow copy)

int n1 = 10;
int n2 = n1;

n2++;
n1 = n1*3;



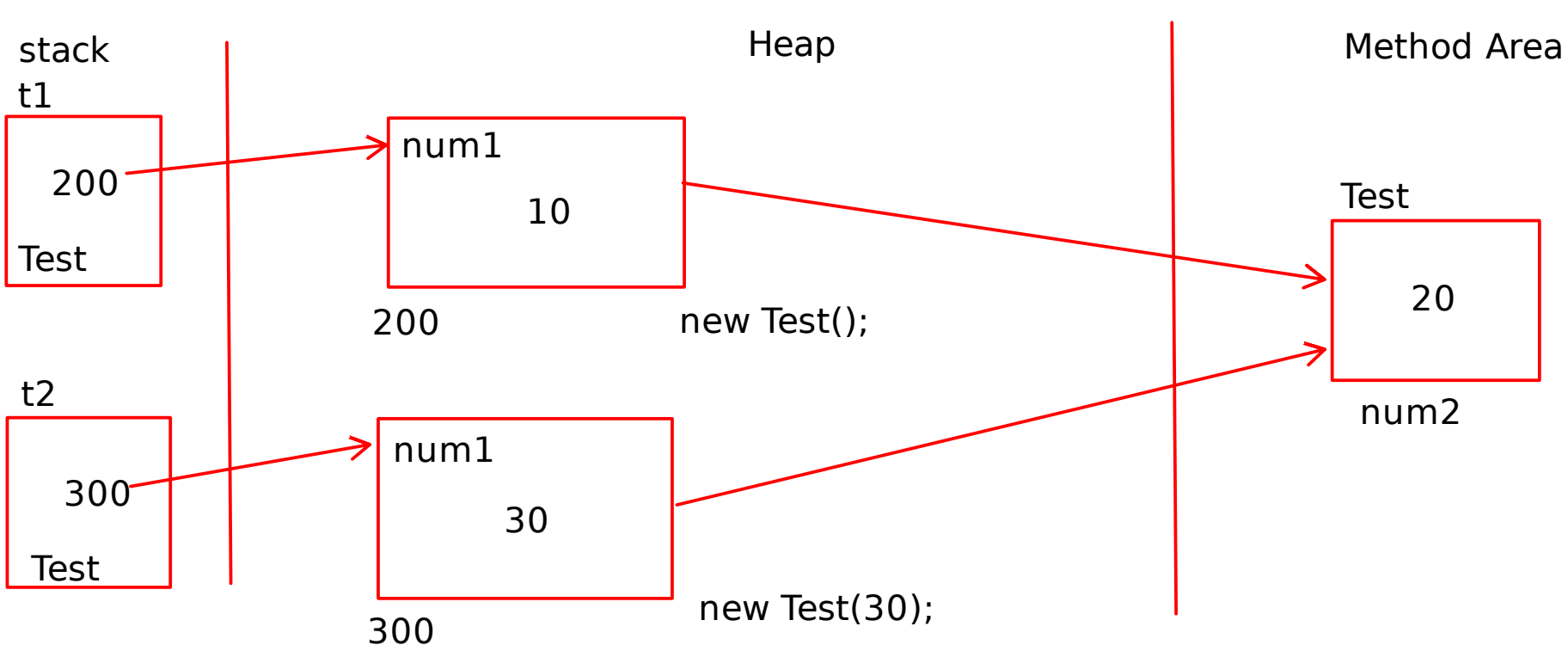
t1

num1
10
num2
20
num3
30
num4
~~40~~ 400

final
- keyword in java
- we can make
1. Variable as final
2. fields as final

3. Methods as final
4. class as final

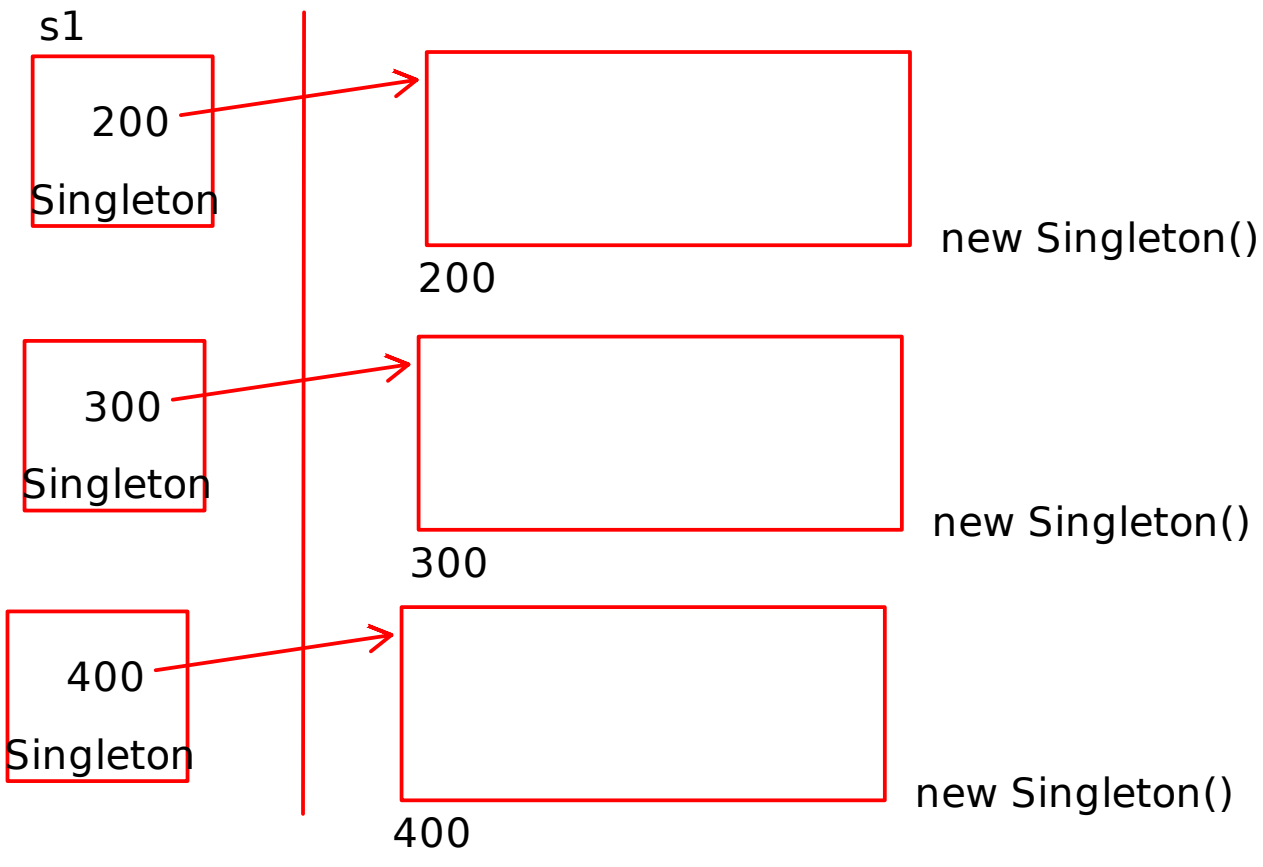
```
const int num1=10;
//num1 = 10;// NOT OK
num1=20;
```

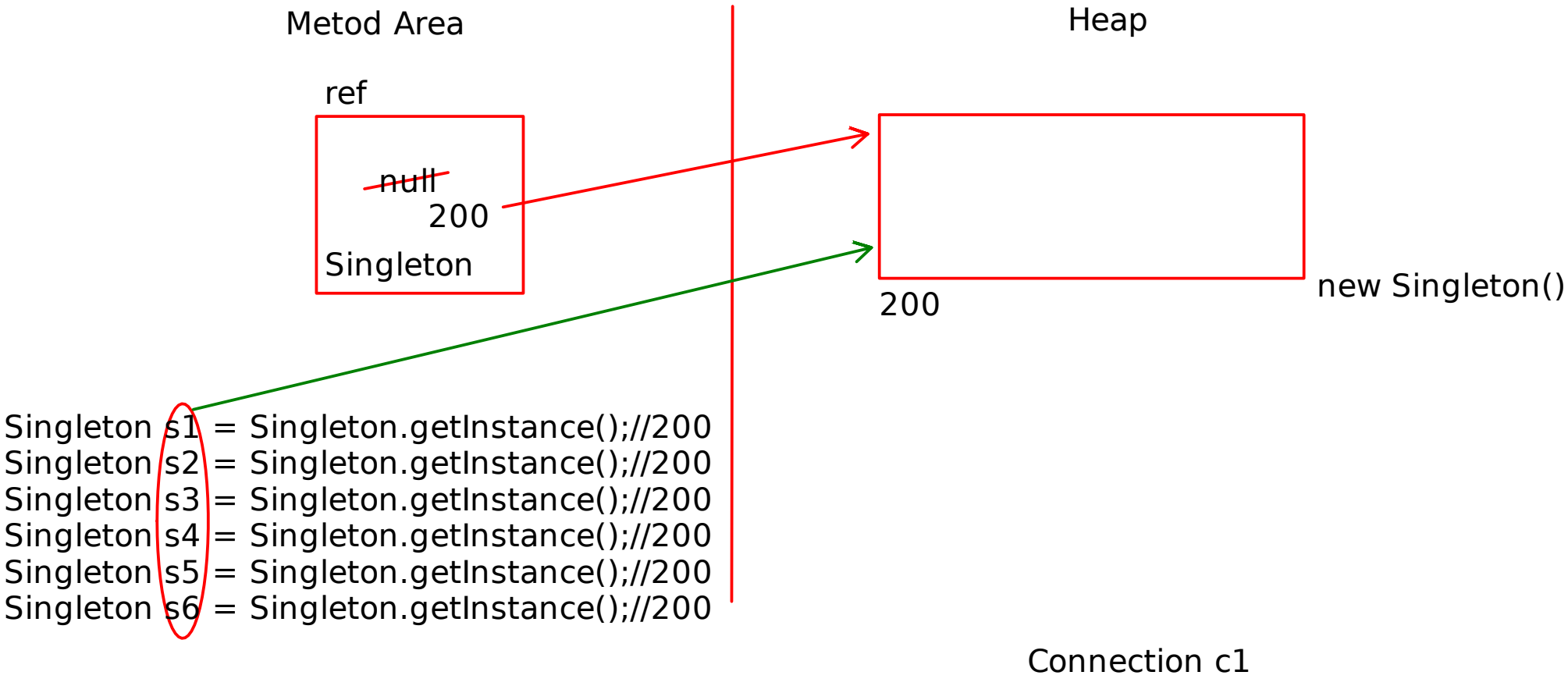


Q. What will be the output

A. 0,0 B 0,1
C. 1,0 D.1,1
E. 1,2 F. 2,2

Singleton Design Pattern






```
class Singleton{
private:
static Singleton *ptr;

Singleton(){
}

public:
static Singleton* getInstance(){
        if(ptr == NULL)
                ptr = new Singleton();
        return ptr;
}

~Singleton(){
delete ptr;
ptr=NULL;
}
};

 Singleton *Singleton::ptr = NULL;
```