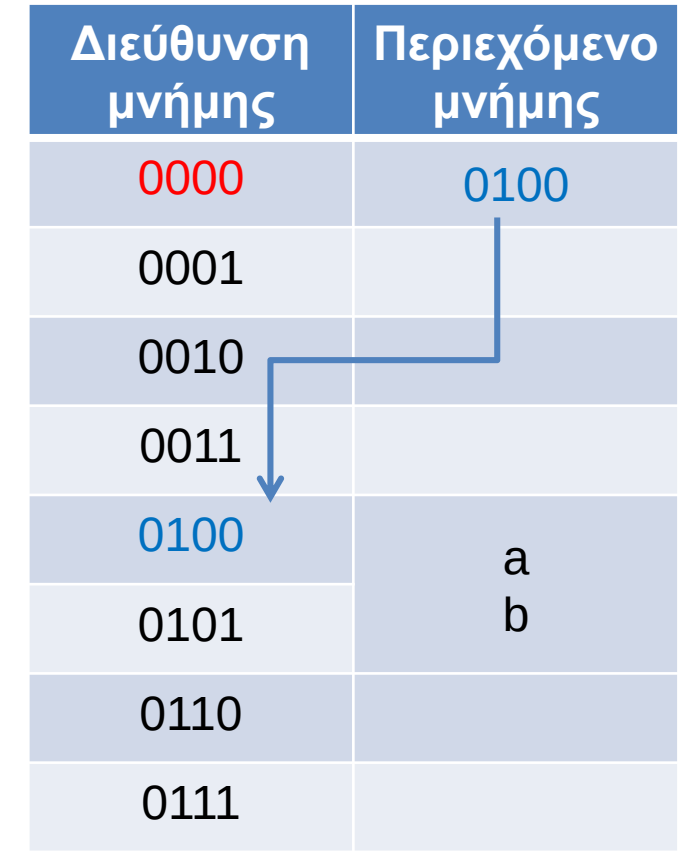
ΤΕΧΝΙΚΕΣ ΑΝΤΙΚΕΙΜΕΝΟΣΤΡΑΦΟΥΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ

Αναφορές
Στοίβα και Σωρός μνήμης
Αντικείμενα παράμετροι
String Interning

Αποθήκευση αντικειμένων

- Οι θέσεις μνήμης των αντικειμένων κρατάνε μια διεύθυνση στο χώρο στον οποίο αποθηκεύεται το αντικείμενο
- Η διεύθυνση αυτή λέγεται αναφορά.
- Οι αναφορές είναι παρόμοιες με τους δείκτες σε άλλες γλώσσες προγραμματισμού με τη διαφορά ότι η Java δεν μας αφήνει να πειράξουμε τις διευθύνσεις.
  - Εμείς χρησιμοποιούμε μόνο τη μεταβλητή του αντικειμένου, όχι το περιεχόμενο της
- Το dereferencing το κάνει η Java αυτόματα.

```
String s = "ab";
```



Παράδειγμα - πινάκες

```
int[] A;
A = new int[2];
A[0] = 10;
A = new int[3];
```

Διεύθυνση μνήμης	Περιεχόμενο μνήμης
0000	
0001	
0010	
0011	
0100	
0101	
0110	
0111	

Παράδειγμα - πινάκες

```
int[] A;
A = new int[2];
A[0] = 10;
A = new int[3];
```

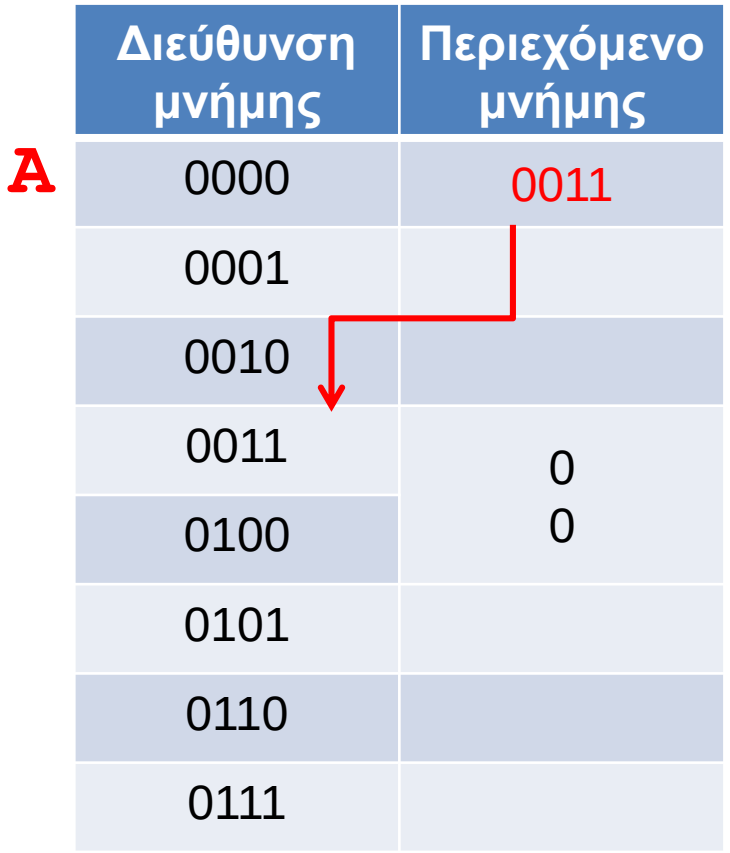
	Διεύθυνση μνήμης	Περιεχόμενο μνήμης
A	0000	null
	0001	
	0010	
	0011	
	0100	
	0101	
	0110	
	0111	

Η δεσμευμένη λέξη null σημαίνει μια κενή αναφορά (μια διεύθυνση που δεν δείχνει πουθενά)

Παράδειγμα - πινάκες

```
int[] A;
A = new int[2];
A[0] = 10;
A = new int[3];
```

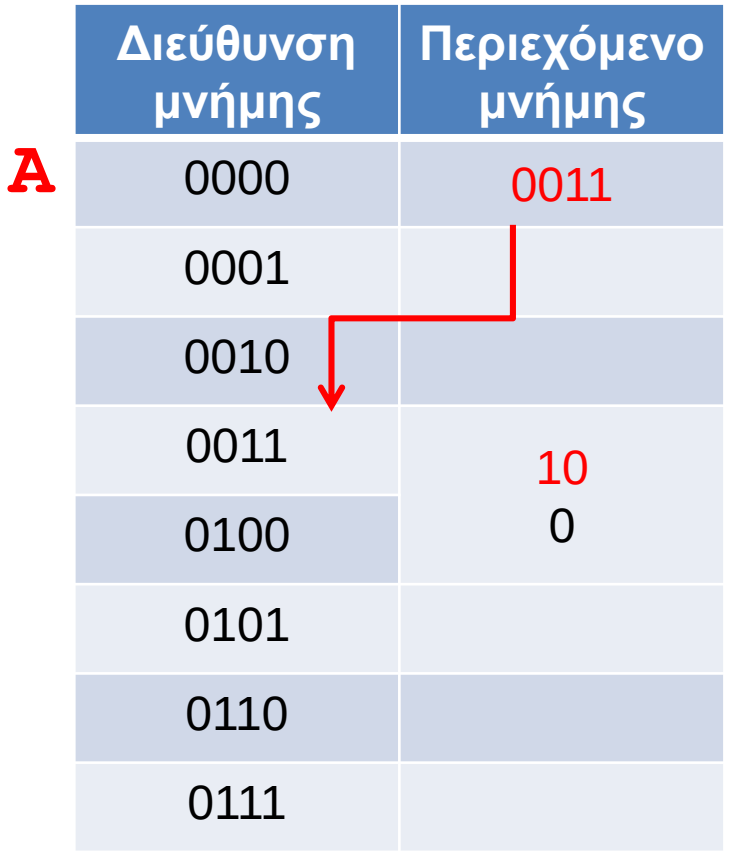
Με την εντολή new δεσμεύουμε δύο θέσεις ακεραίων και η αναφορά του A δείχνει σε αυτό το χώρο που δεσμεύσαμε



Παράδειγμα - πινάκες

```
int[] A;
A = new int[2];
A[0] = 10;
A = new int[3];
```

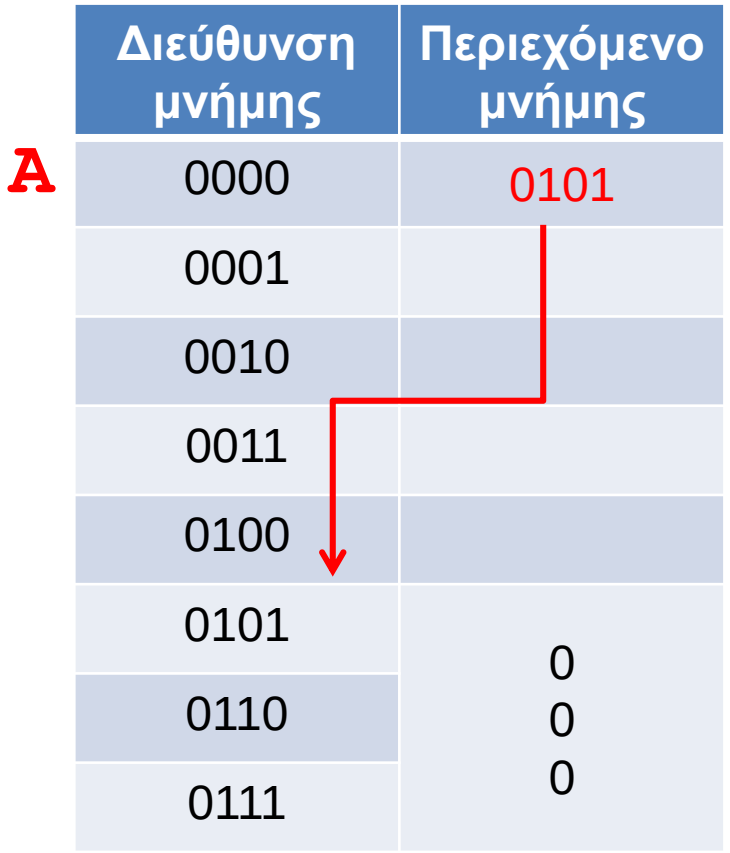
Ο τελεστής [] για τον πίνακα μας πάει στην αντίστοιχη θέση του χώρου που κρατήσαμε



Παράδειγμα - πινάκες

```
int[] A;
A = new int[2];
A[0] = 10;
A = new int[3];
```

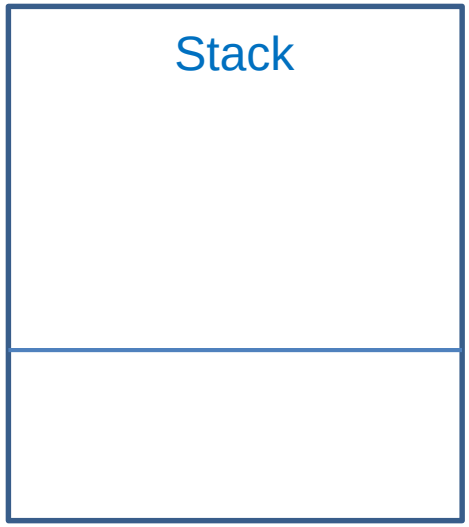
Με νέα κλήση της new δεσμεύουμε νέο χώρο για το Α, και αν δεν έχουμε κρατήσει την προηγούμενη αναφορά σε κάποια άλλη μεταβλητή τότε χάνεται (garbage collection), όπως και οι τιμές που είχαμε αποθηκεύσει στον πίνακα.

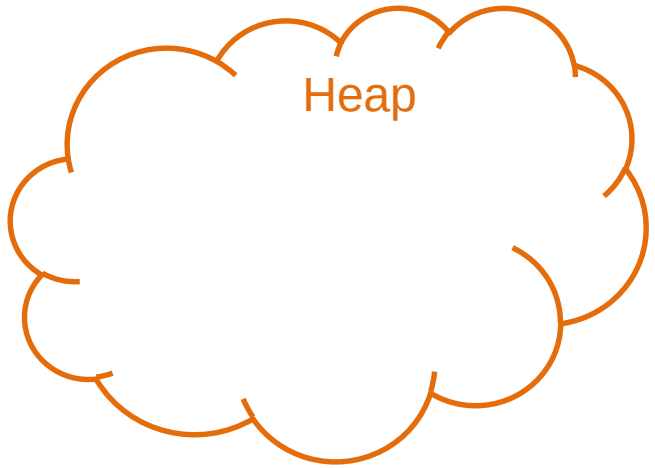


ΣΤΟΙΒΑ ΚΑΙ ΣΩΡΟΣ

Διαχείριση μνήμης από το JVM

- Η μνήμη χωρίζεται σε δύο τμήματα
  - Τη στοίβα (stack) που χρησιμοποιείται για να κρατάει πληροφορία για τις τοπικές μεταβλητές κάθε μεθόδου/block.
  - Το σωρό (heap) που χρησιμοποιείται για να δεσμεύουμε μνήμη για τα αντικείμενα



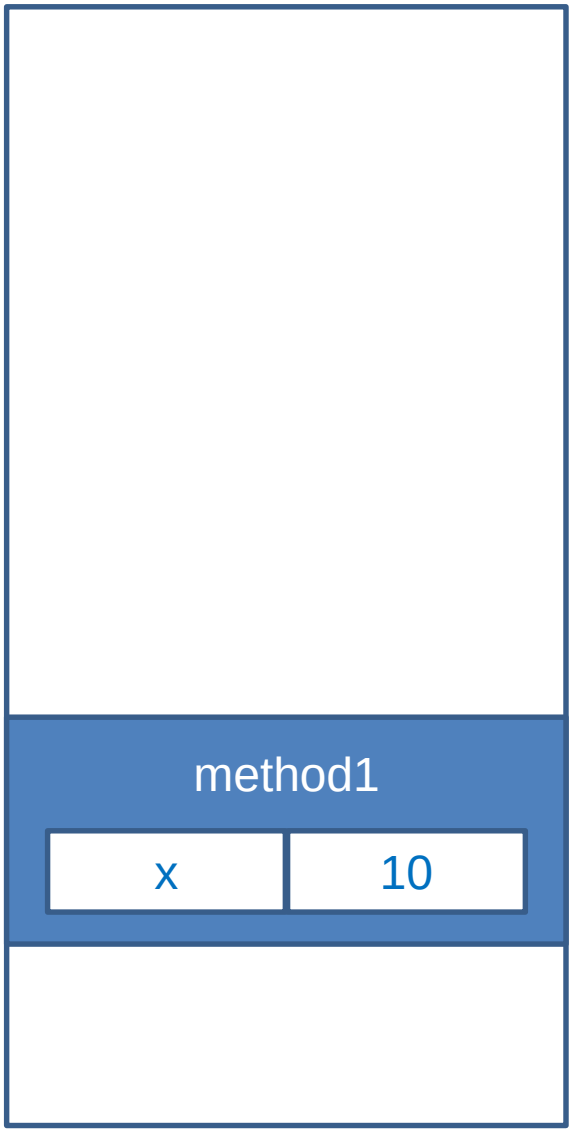


Stack

- Κάθε φορά που καλείται μία μέθοδος, δημιουργείται ένα «πλαίσιο» (frame) για την μέθοδο στη στοίβα
  - Δημιουργείται ένας χώρος μνήμης που αποθηκεύει τις παραμέτρους και τις τοπικές μεταβλητές της μεθόδου.
- Αν η μέθοδος καλέσει μία άλλη μέθοδο θα δημιουργηθεί ένα νέο πλαίσιο και θα τοποθετηθεί (push) στην κορυφή της στοίβας.
- Όταν βγούμε από την μέθοδο το πλαίσιο αφαιρείται (pop) από την κορυφή της στοίβας και επιστρέφουμε στην προηγούμενη μέθοδο
- Στη βάση της στοίβας είναι η μέθοδος main.

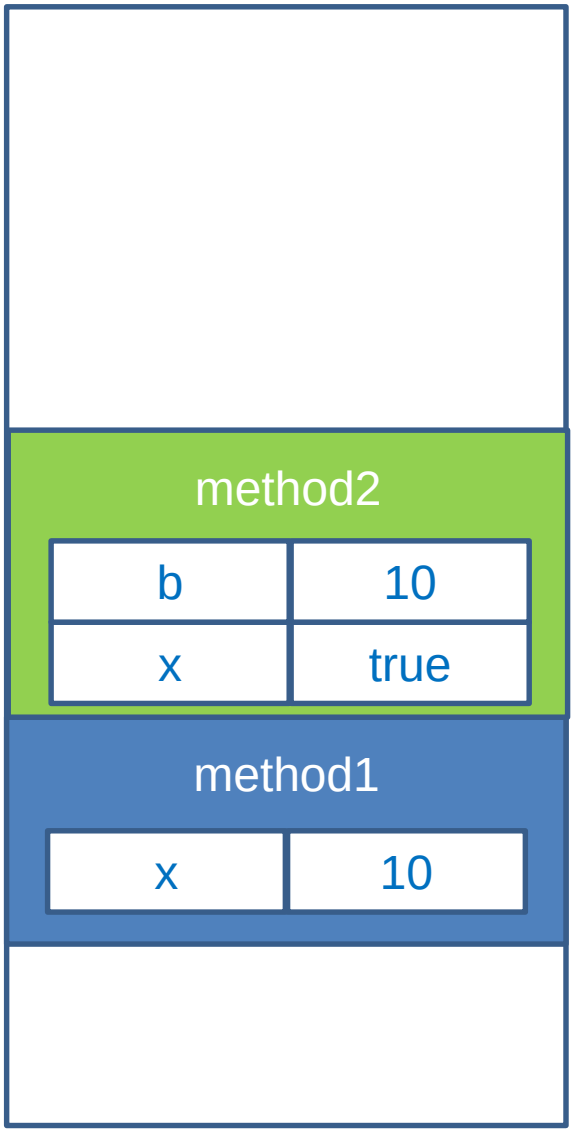
Παράδειγμα

```
public void method1(){
   int x = 10;
   method2(x);
}
```



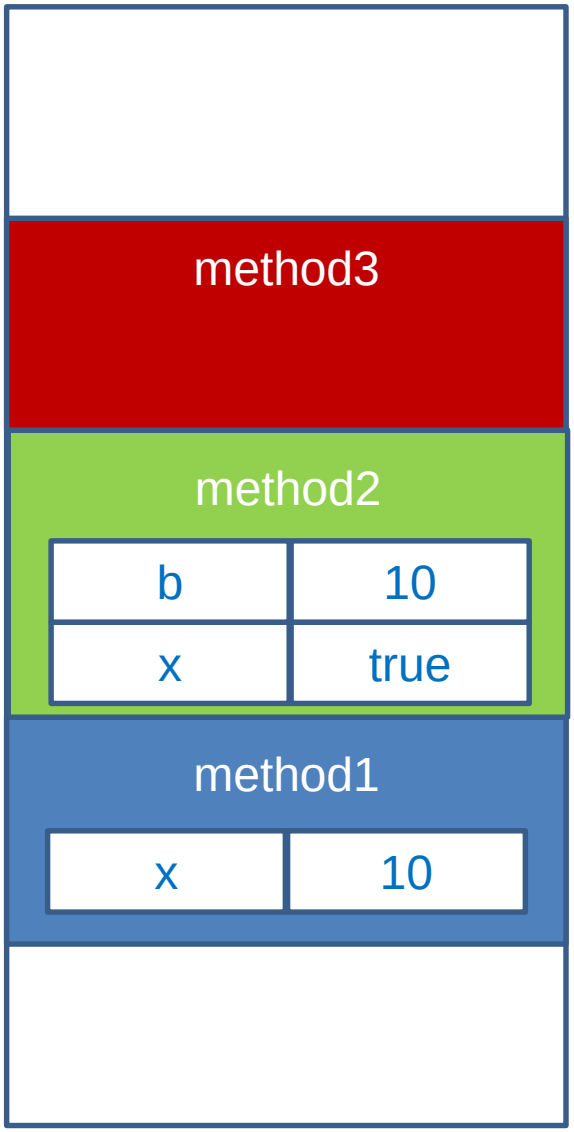
Παράδειγμα

```
public void method1(){
   int x = 10;
   method2(x);
}

public void method2(int b){
   boolean x = true;
   method3();
}
```



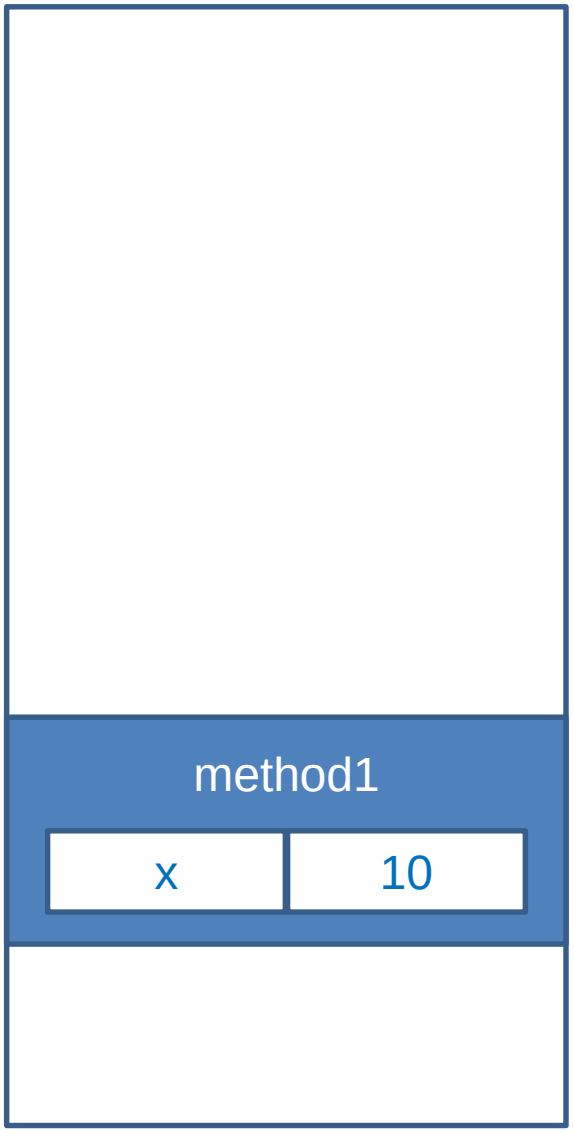
Παράδειγμα

```
public void method1(){
   int x = 10;
   method2(x);
}

public void method2(int b){
   boolean x = true;
   method3();
}

public void method3()
{…}
```



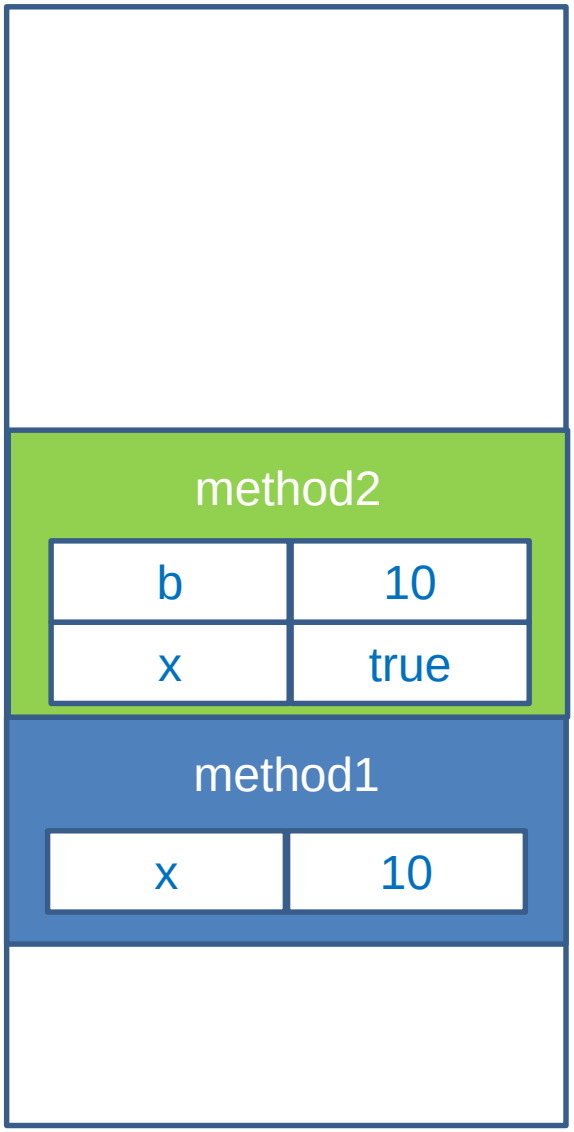
Παράδειγμα

```
public void method1(){
   int x = 10;
   method2(x);
   method3();
}
```



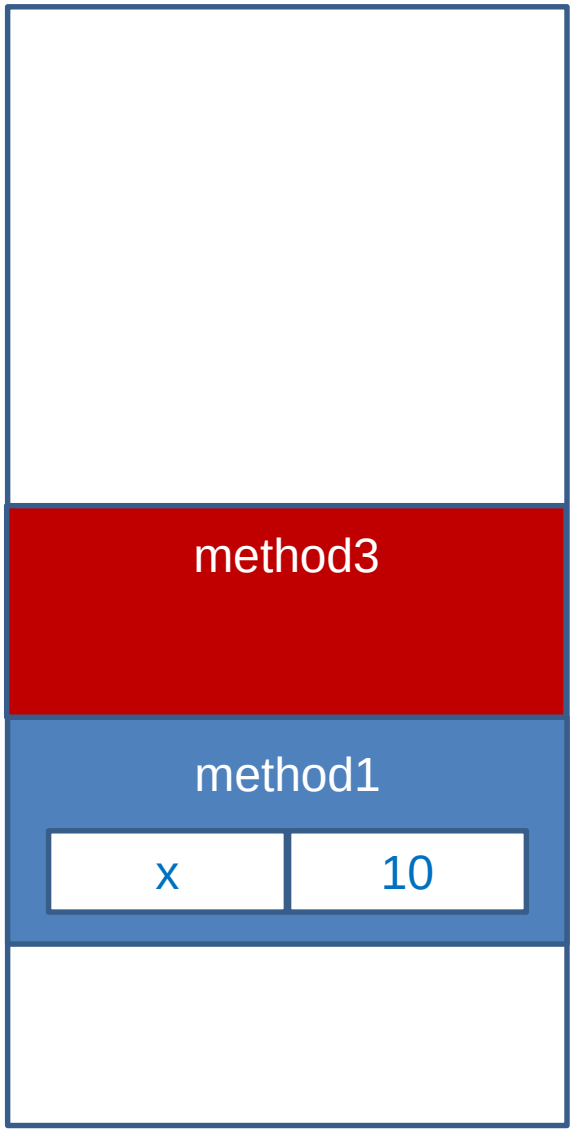
Παράδειγμα

```
public void method1(){
   int x = 10;
   method2(x);
   method3()
}

public void method2(int b){
   boolean x = (b==10);
   ...
}
```



Παράδειγμα

```
public void method1(){
   int x = 10;
   method2(x);
}

public void method3()
{…}
```



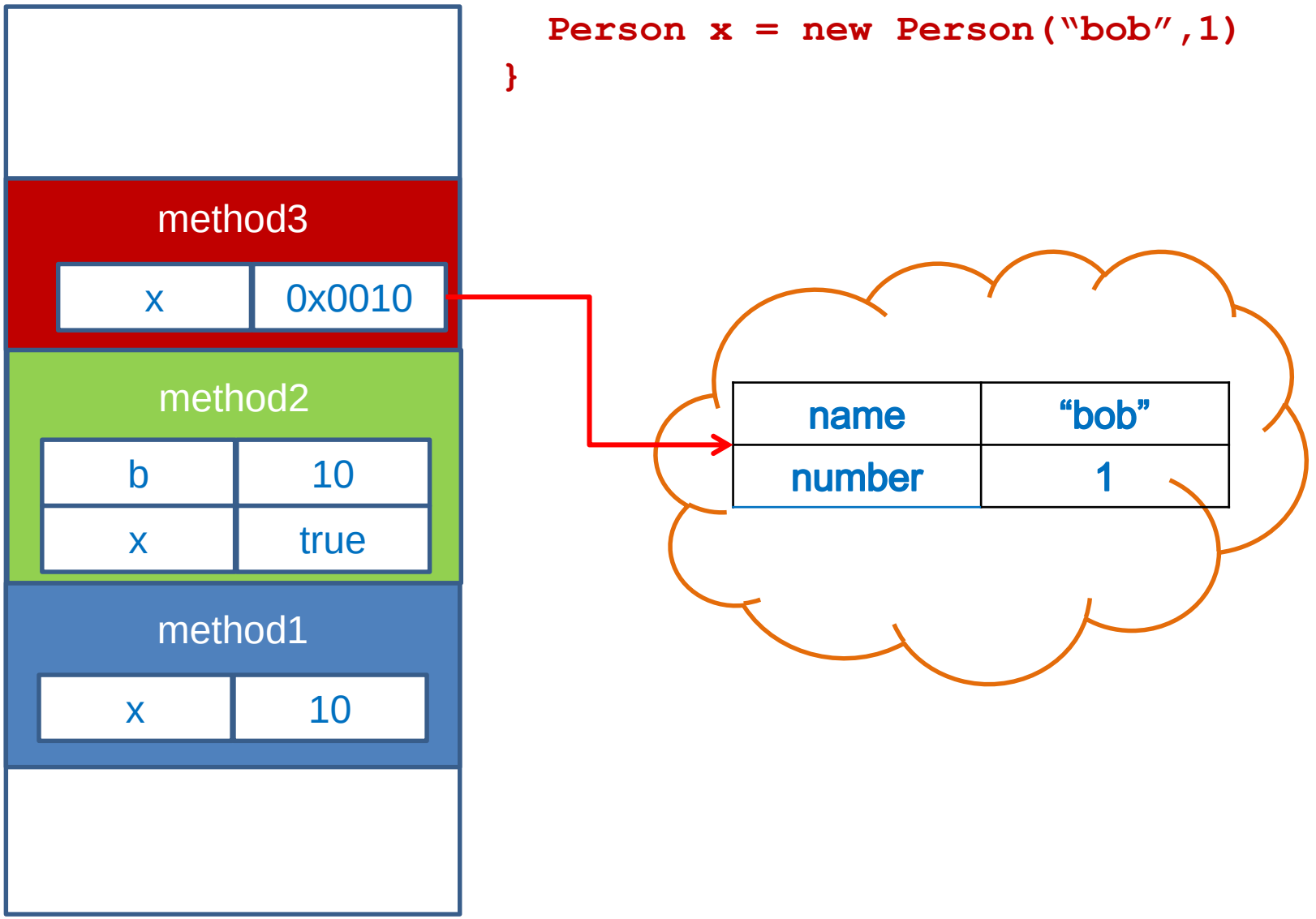
Heap

- Όταν μέσα σε μία μέθοδο δημιουργούμε ένα αντικείμενο με την new γίνονται τα εξής
  - στο πλαίσιο (frame) της μεθόδου (στη στοίβα) υπάρχει μια τοπική μεταβλητή που κρατάει την αναφορά στο αντικείμενο
  - Η κλήση της new δεσμεύει χώρο μνήμης στο σωρό (heap) για να κρατήσει τα πεδία του αντικειμένου.
  - Η αναφορά δείχνει στη θέση μνήμης που δεσμεύτηκε.

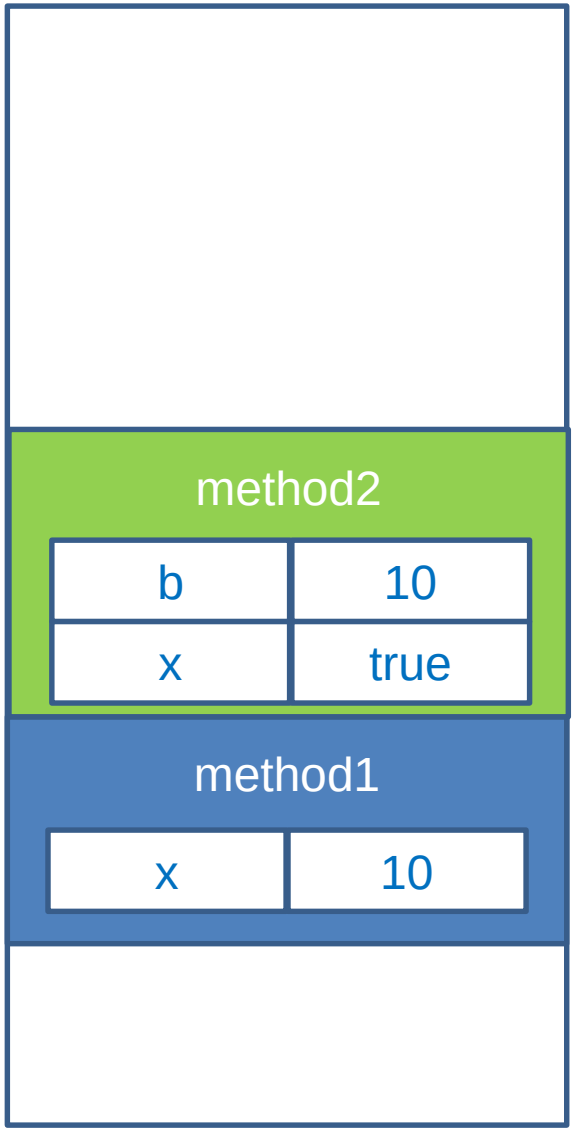
```
public class Person
{
    private String name;
    private int number;

    public Person(String initName, int initNumber){
        name = initName;
        number = initNumber;
    }

    public void set(String name, int number){
        this.name = name;
        this.number = number;
    }

    public String toString( ){
        return (name + " " + number);
    }
}
```

Παράδειγμα

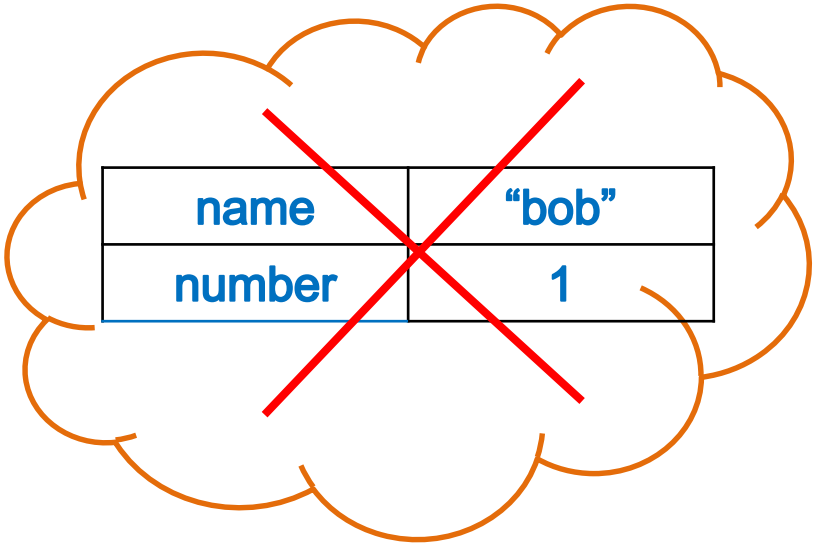
```
public void method3()
{
   Person x = new Person("bob",1)
}
```



Παράδειγμα



Όταν επιστρέφουμε από την μέθοδο method3 η αναφορά προς το αντικείμενο Person παύει να υπάρχει.

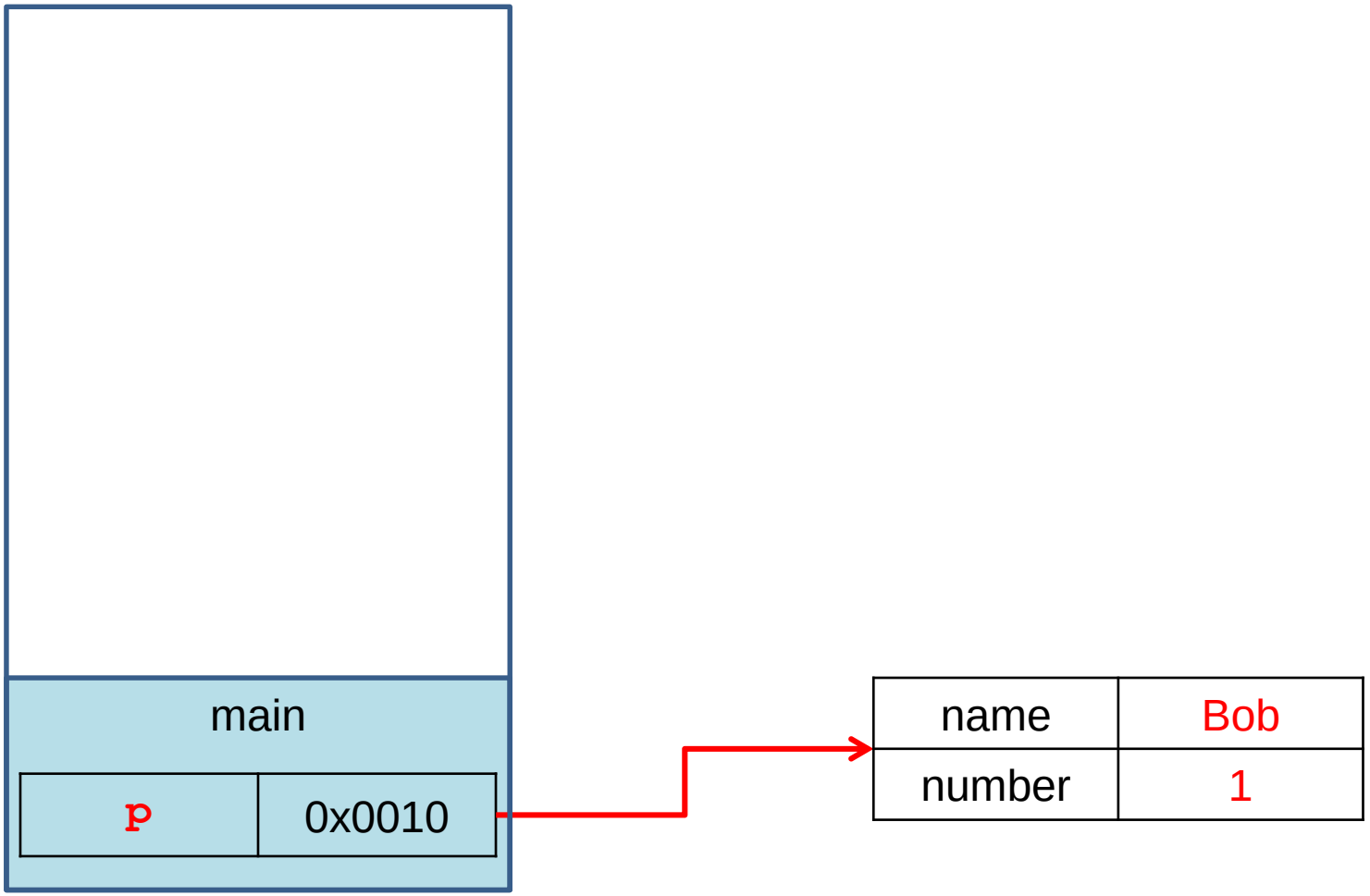


Αν δεν υπάρχουν άλλες αναφορές στο αντικείμενο τότε ο garbage collector αποδεσμεύει τη μνήμη του αντικειμένου

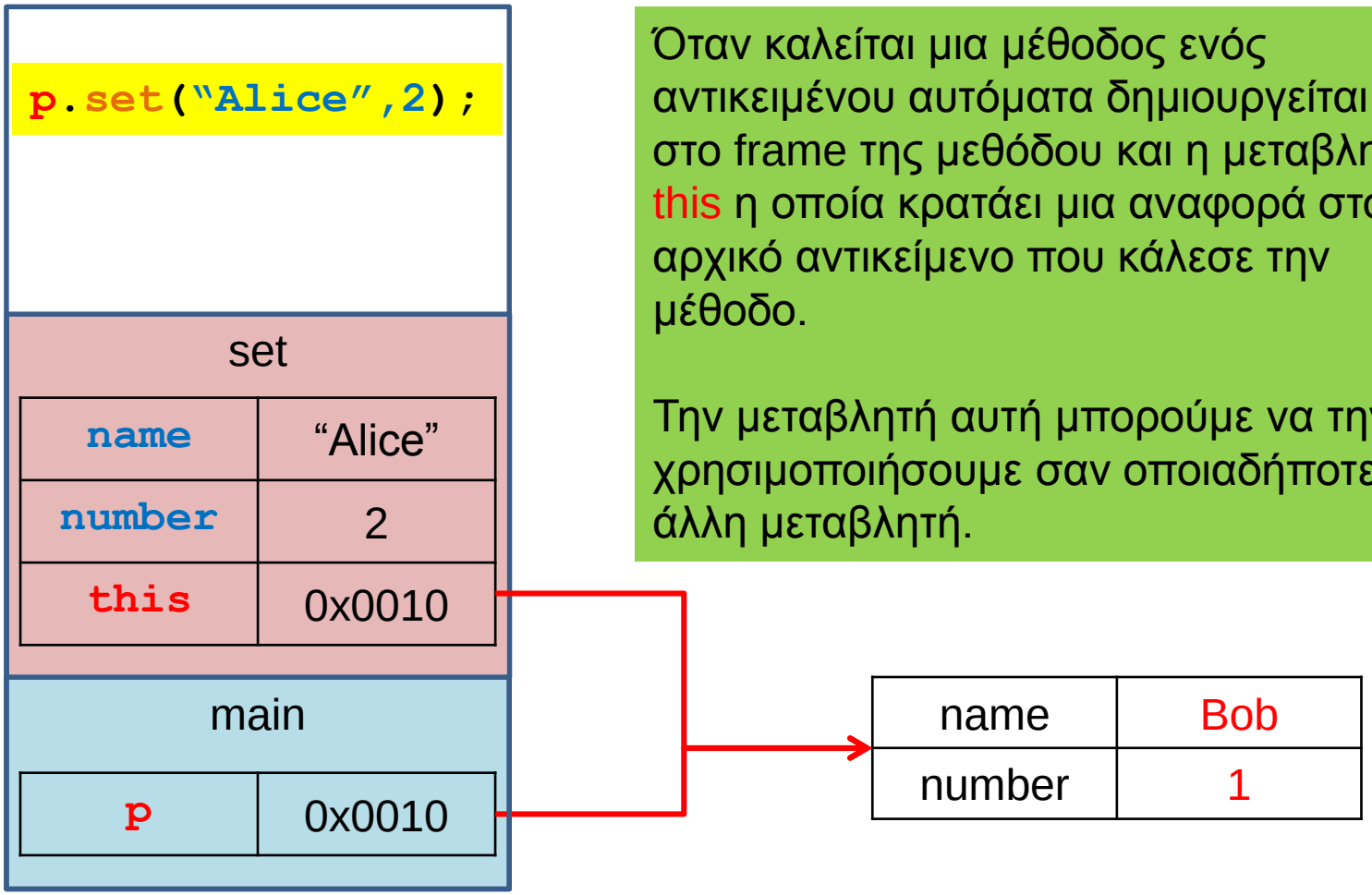
Κλήση μεθόδου από αντικείμενο

```
public class ObjectMethodCallDemo
{
    public static void main(String[] args)
    {
        Person p = new Person("Bob", 1);
        p.set("Alice",2);
        System.out.println(p);
    }
}
```

Εξέλιξη του προγράμματος



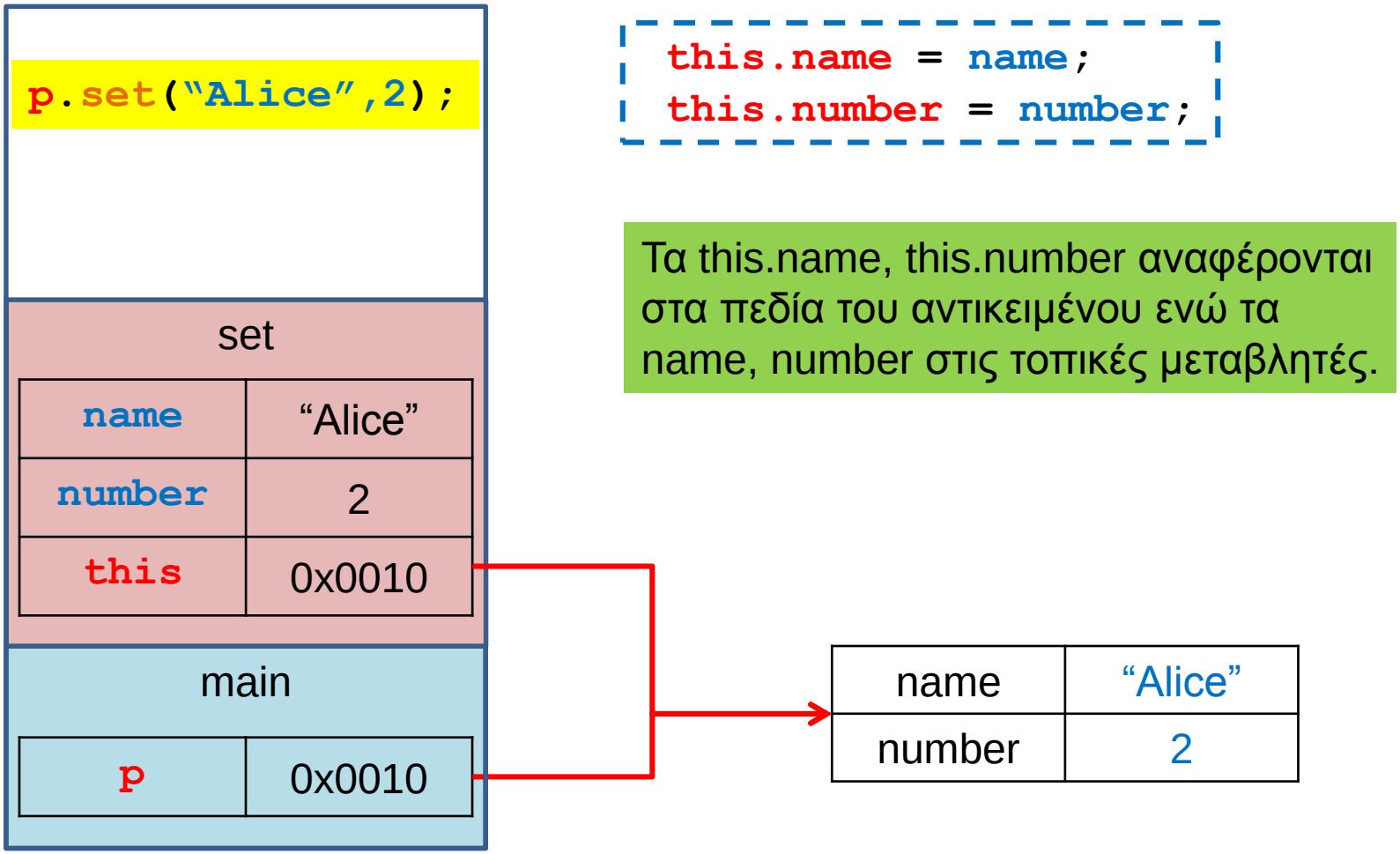
Εξέλιξη του προγράμματος



Όταν καλείται μια μέθοδος ενός αντικειμένου αυτόματα δημιουργείται στο frame της μεθόδου και η μεταβλητή this η οποία κρατάει μια αναφορά στο αρχικό αντικείμενο που κάλεσε την μέθοδο.

Την μεταβλητή αυτή μπορούμε να την χρησιμοποιήσουμε σαν οποιαδήποτε άλλη μεταβλητή.

Εξέλιξη του προγράμματος



Αντικείμενα ως παράμετροι

- Όταν περνάμε παραμέτρους σε μία μέθοδο το πέρασμα γίνεται πάντα δια τιμής (call-by-value)
  - Δηλαδή απλά περνάμε τα περιεχόμενα της θέσης μνήμης της συγκεκριμένης μεταβλητής.
  - Για μεταβλητές πρωταρχικού τύπου, αλλαγές στην τιμή της παραμέτρου δεν αλλάζουν την μεταβλητή που περάσαμε σαν όρισμα.

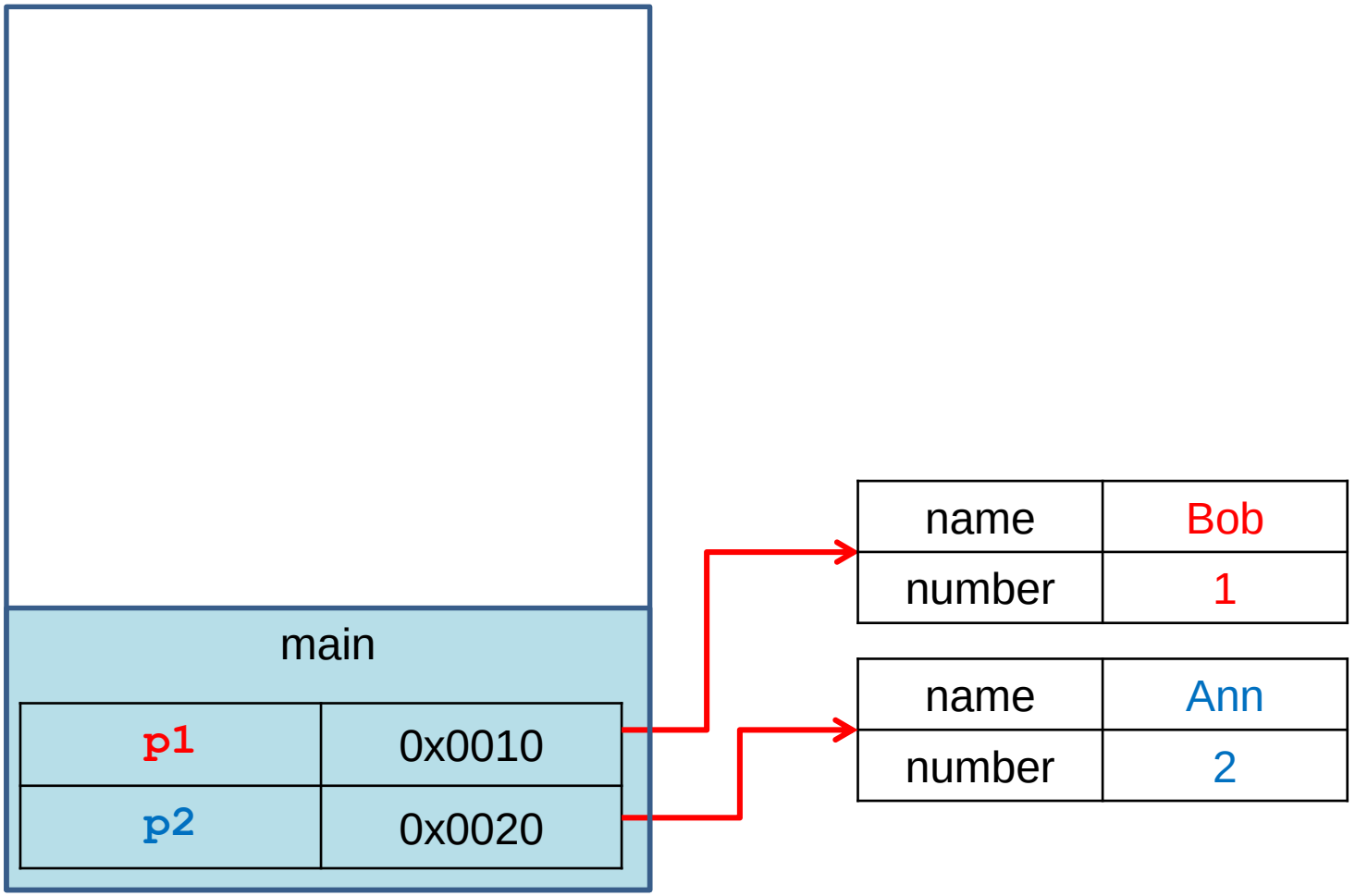
- Τι γίνεται όμως αν η παράμετρος είναι ένα αντικείμενο?
  - Τα περιεχόμενα της θέσης μνήμης μιας μεταβλητής-αντικείμενο είναι μια αναφορά.
  - Αν μέσα στην μέθοδο αλλάξουν τα περιεχόμενα του αντικειμένου (εκεί που δείχνει η αναφορά) τότε αλλάζει και η μεταβλητή-αντικείμενο που περάσαμε.

```
public class Person
{
    private String name;
    private int number;

    public Person(String initName, int initNumber){
        name = initName;
        number = initNumber;
    }

    public void set(String newName, int newNumber){
        name = newName;
        number = newNumber;
    }

    public void copier(Person other) {
        other.name = name;
        other.number = number;
    }

    public String toString( ){
        return (name + " " + number);
    }
}
```

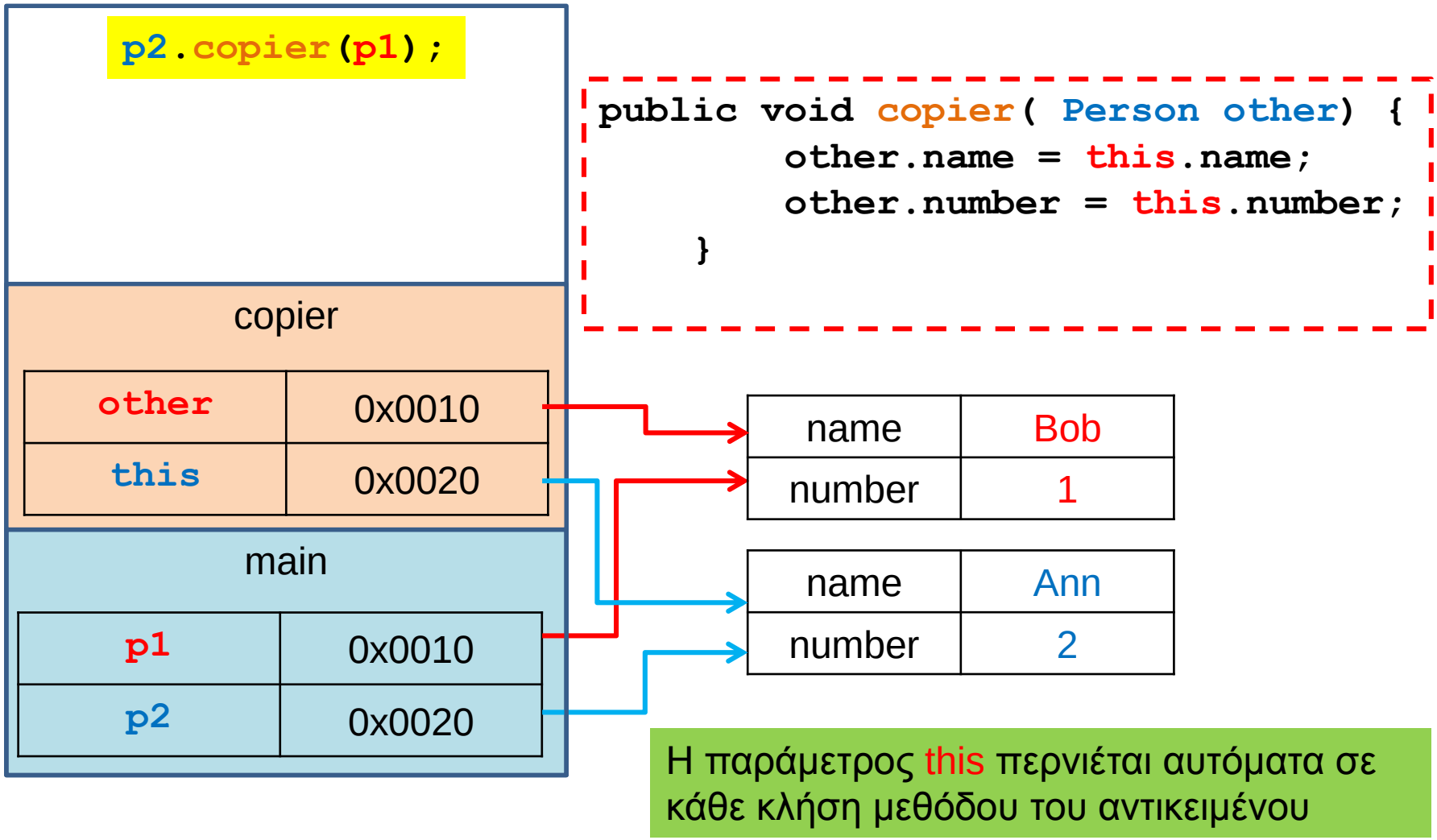
Παράδειγμα

```
public class ClassParameterDemo
{
    public static void main(String[] args)
    {
        Person p1 = new Person("Bob", 1);
        Person p2 = new Person("Ann", 2);
        p2.copier(p1);
        System.out.println(p1);
     }
}
```

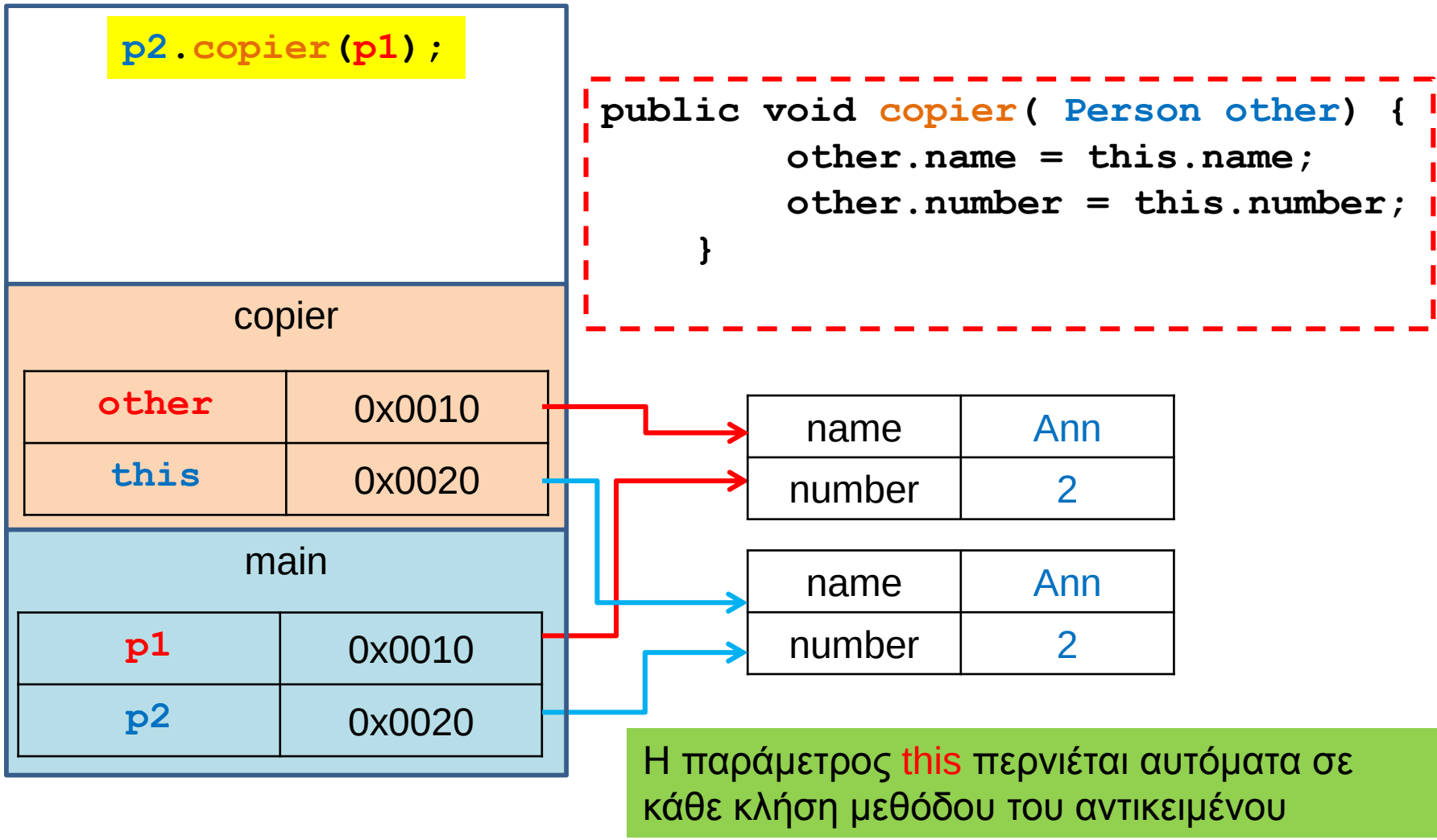
Εξέλιξη του προγράμματος



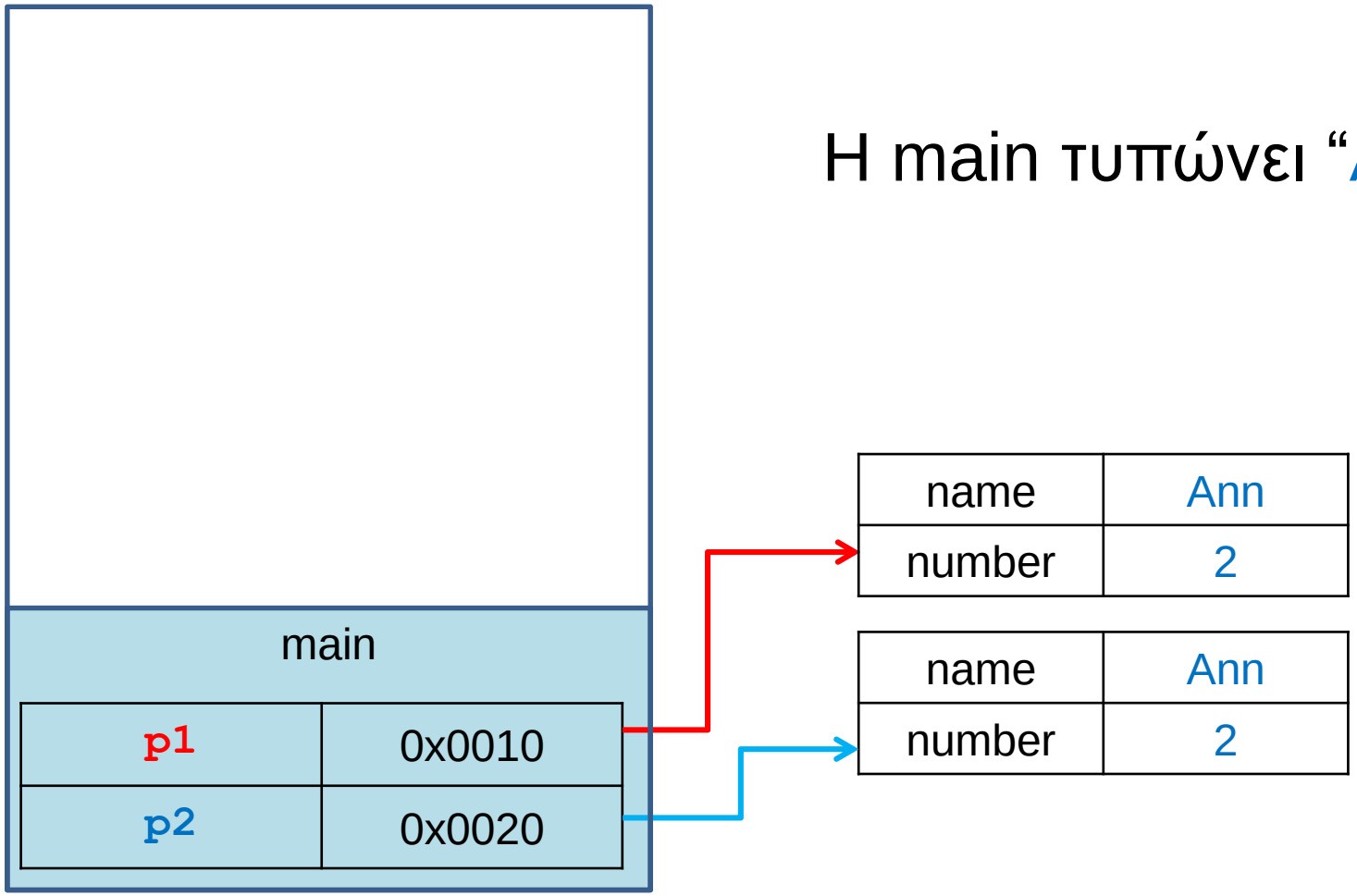
Εξέλιξη του προγράμματος



Εξέλιξη του προγράμματος



Εξέλιξη του προγράμματος



Η main τυπώνει “Ann 2”

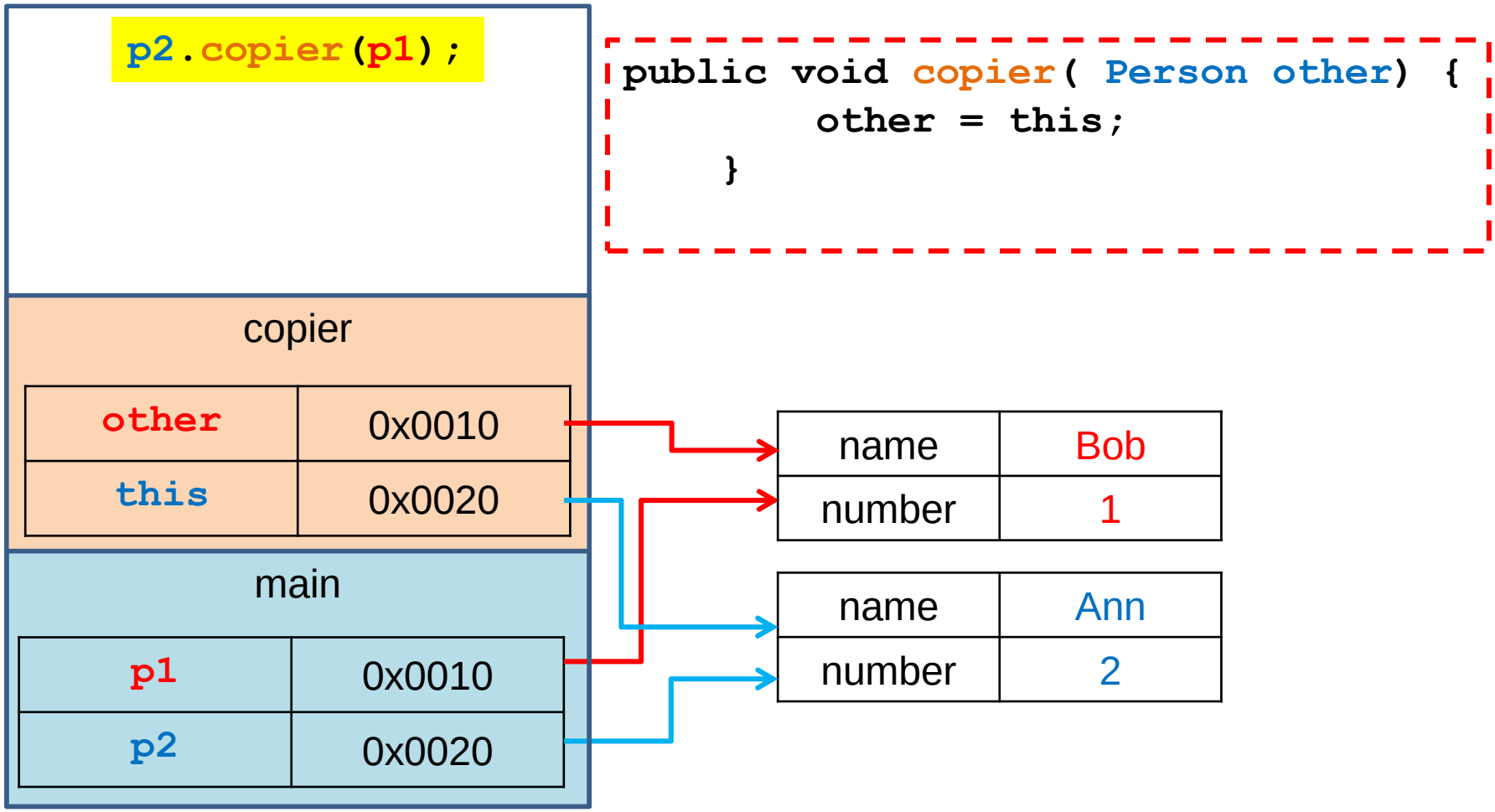
Μια άλλη υλοποίηση της copier

```
public void copier( Person other) {
        other = this;
    }
```

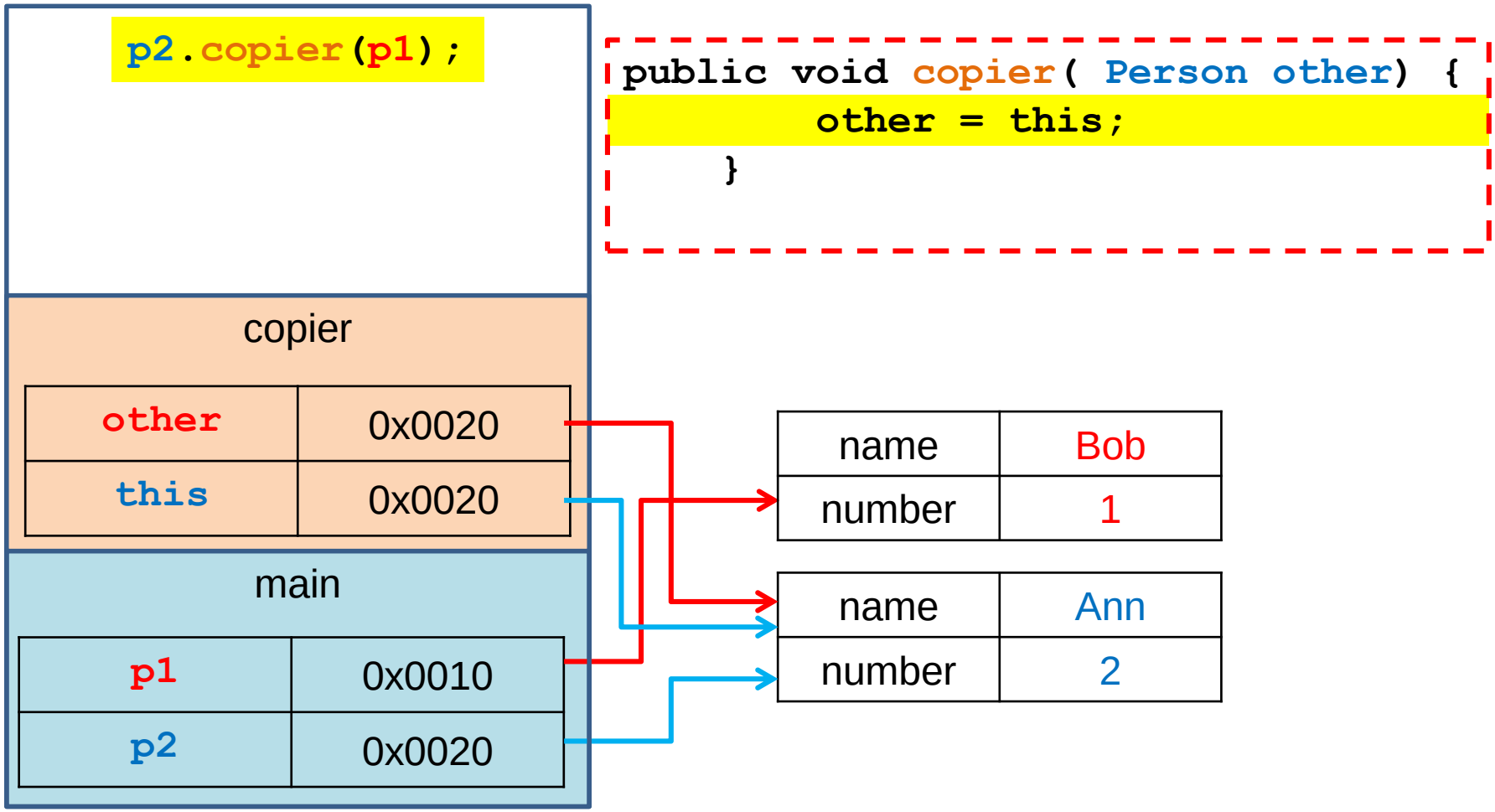
```
public class ClassParameterDemo
{
    public static void main(String[] args)
    {
        Person p1 = new Person("Bob", 1);
        Person p2 = new Person("Ann", 2);
        p2.copier(p1);
        System.out.println(p1);
     }
}
```

Τι θα τυπώσει?

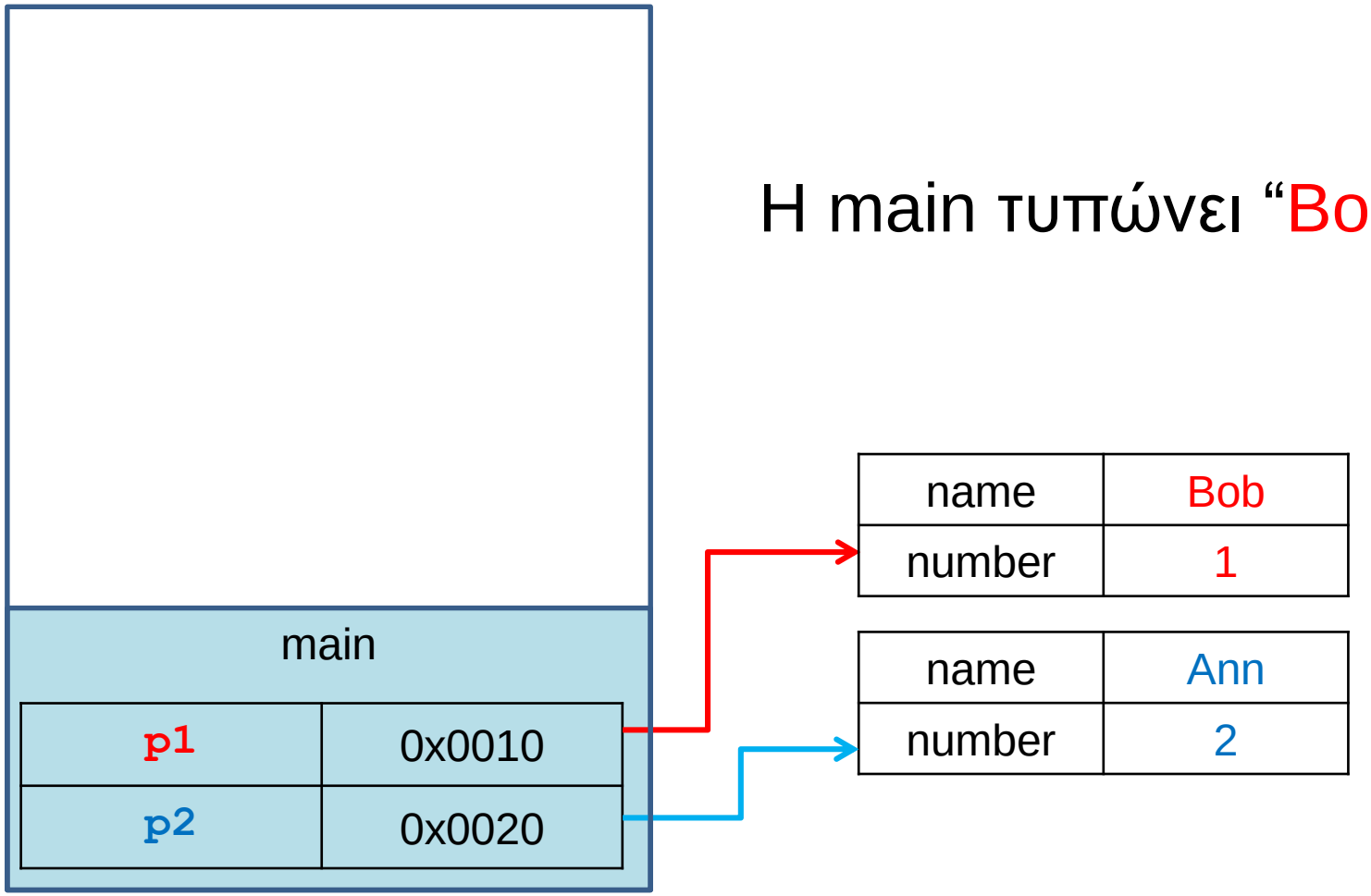
Εξέλιξη του προγράμματος



Εξέλιξη του προγράμματος



Εξέλιξη του προγράμματος



Η main τυπώνει “Bob 1”

Μια ακόμη υλοποίηση της copier

```
public void copier( Person other) {
        other = new Person(this.name, this.number);
    }
```

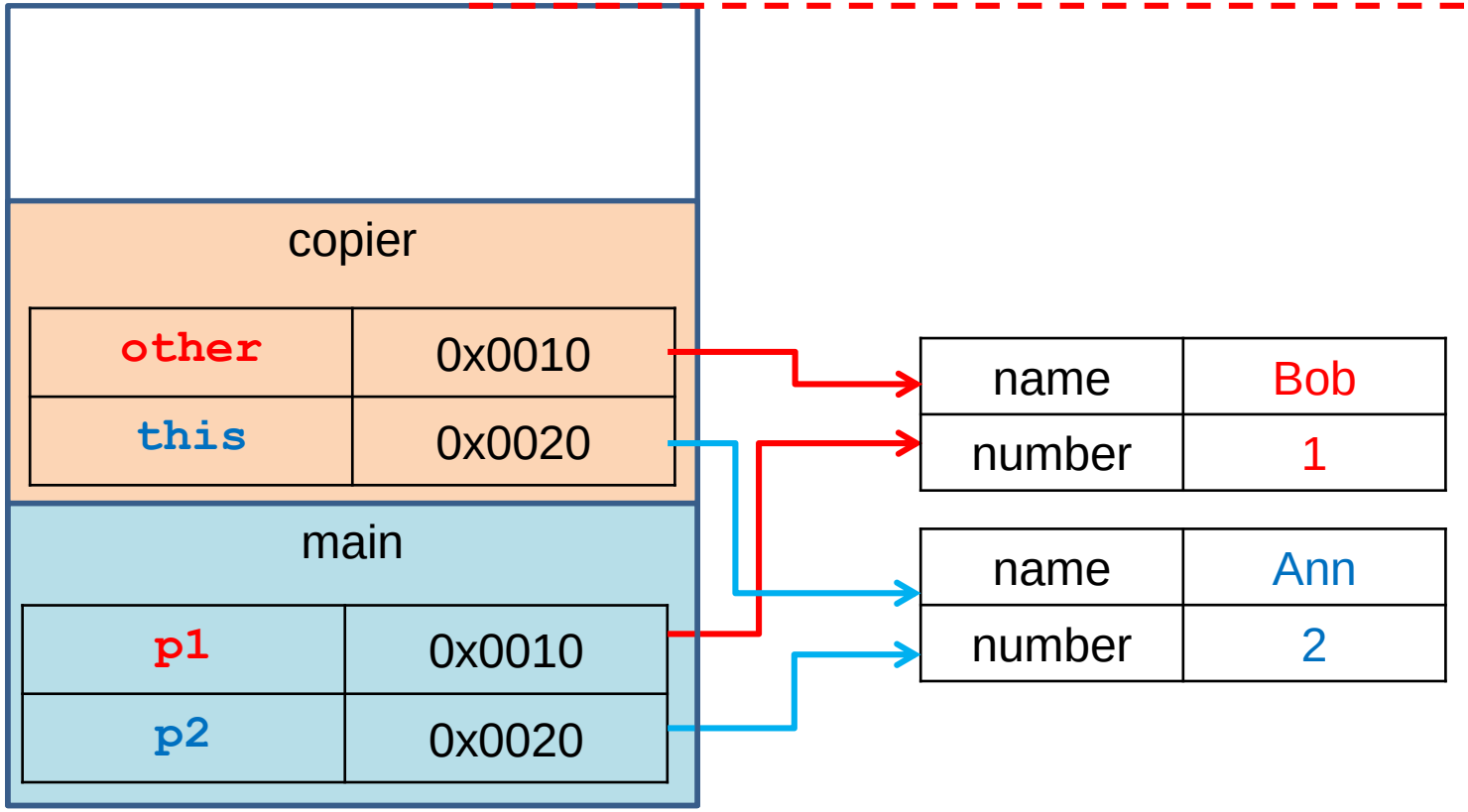
```
public class ClassParameterDemo
{
    public static void main(String[] args)
    {
        Person p1 = new Person("Bob", 1);
        Person p2 = new Person("Ann", 2);
        p2.copier(p1);
        System.out.println(p1);
     }
}
```

Τι θα τυπώσει?

Εξέλιξη του προγράμματος

```
p2.copier(p1);
```

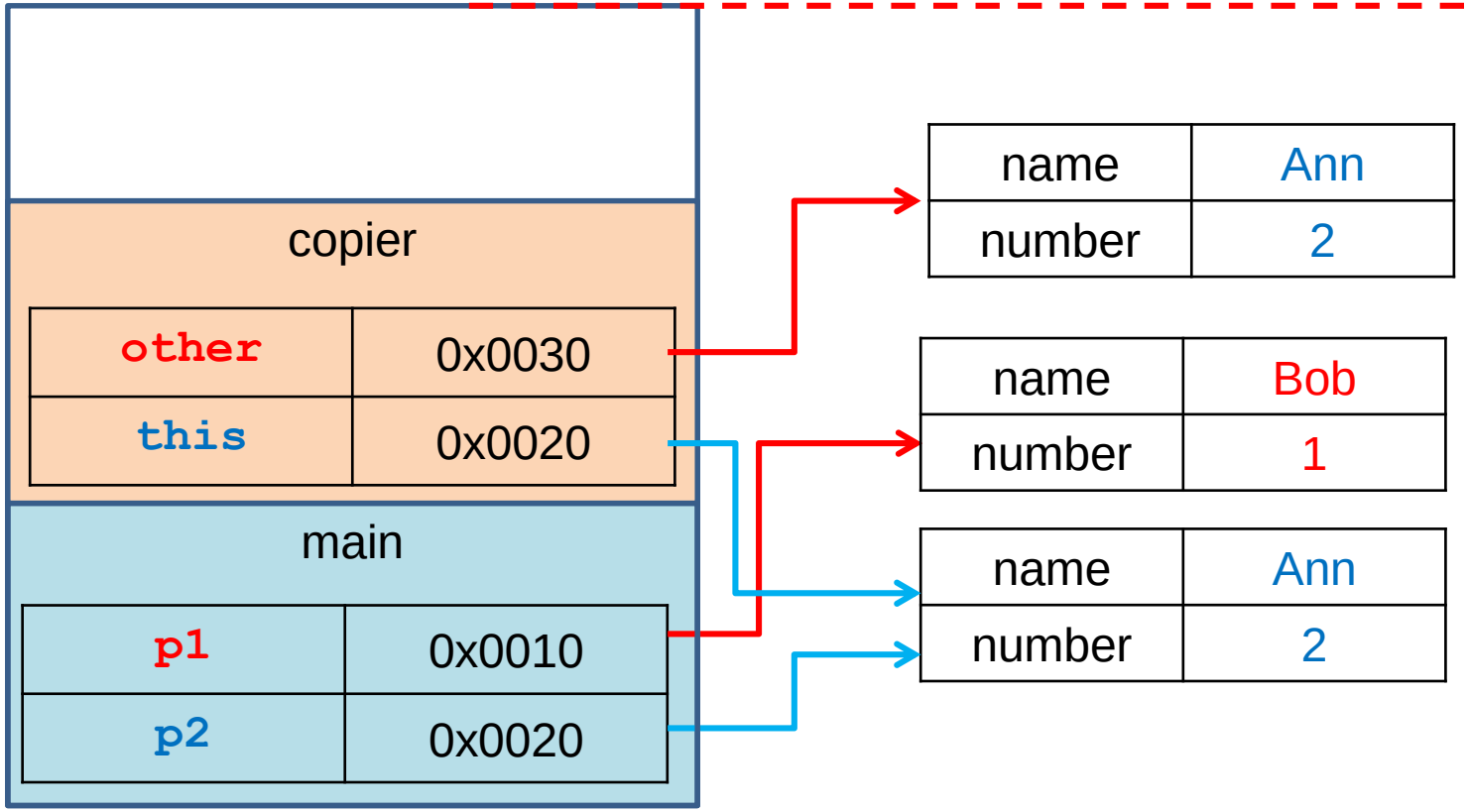
```
public void copier( Person other) {
  other = new Person(this.name, this.number);
}
```



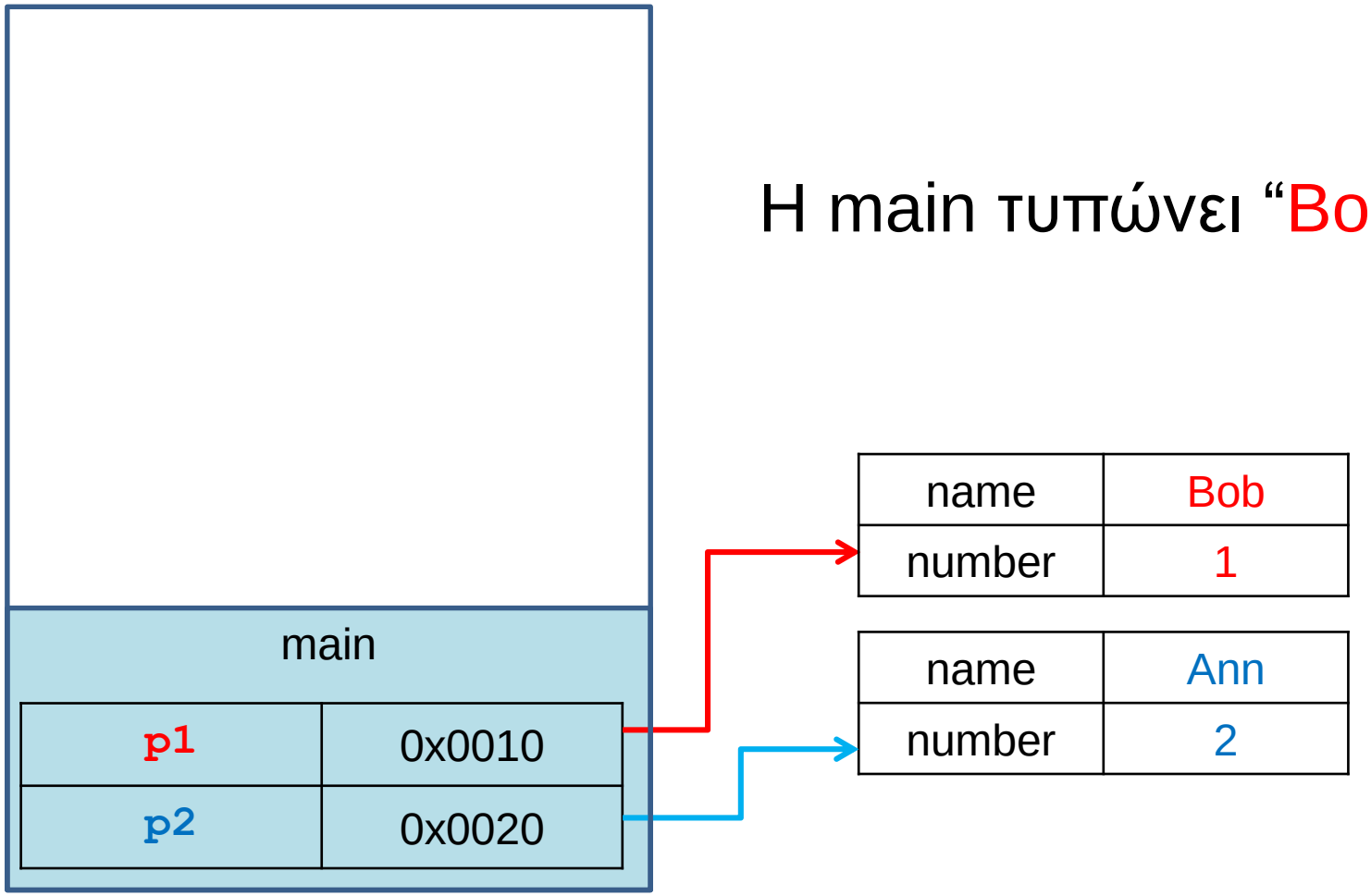
Εξέλιξη του προγράμματος

```
p2.copier(p1);
```

```
public void copier( Person other) {
  other = new Person(this.name, this.number);
}
```



Εξέλιξη του προγράμματος



Η main τυπώνει “Bob 1”

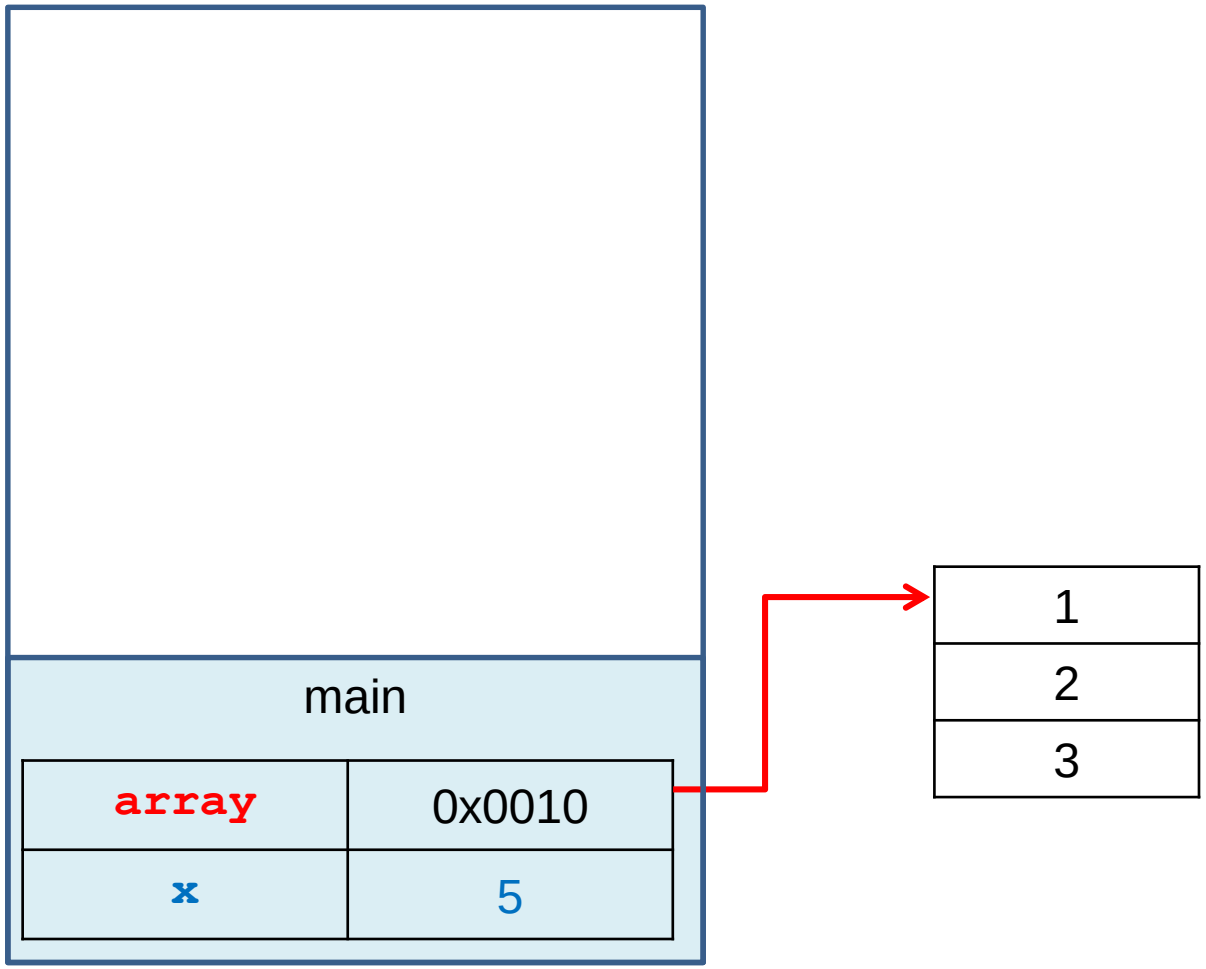
Αλλαγή παραμέτρων

- Στο πρόγραμμα που είδαμε η νέα τιμή του other χάνεται όταν επιστρέφουμε από την συνάρτηση και η p1 παραμένει αμετάβλητη.
- Αυτό γιατί το πέρασμα των παραμέτρων γίνεται κατά τιμή, και η μεταβλητή other είναι τοπική. Ότι αλλαγή κάνουμε στην τιμή της θα έχει εμβέλεια μόνο μέσα στην copier.
  - Το νέο αντικείμενο που δημιουργήσαμε στην περίπτωση αυτή θα χαθεί άμα φύγουμε από τη μέθοδο εφόσον δεν υπάρχει κάποια αναφορά σε αυτό.
- Η αλλαγή στην τιμή της other είναι διαφορετική από την αλλαγή στα περιεχόμενα της διεύθυνσης στην οποία δείχνει η other
  - Οι αλλαγές στα περιεχόμενα αλλάζουν τον χώρο μνήμης στο σωρό (heap). Οι αλλαγές επηρεάζουν όλες τις αναφορές στο αντικείμενο.

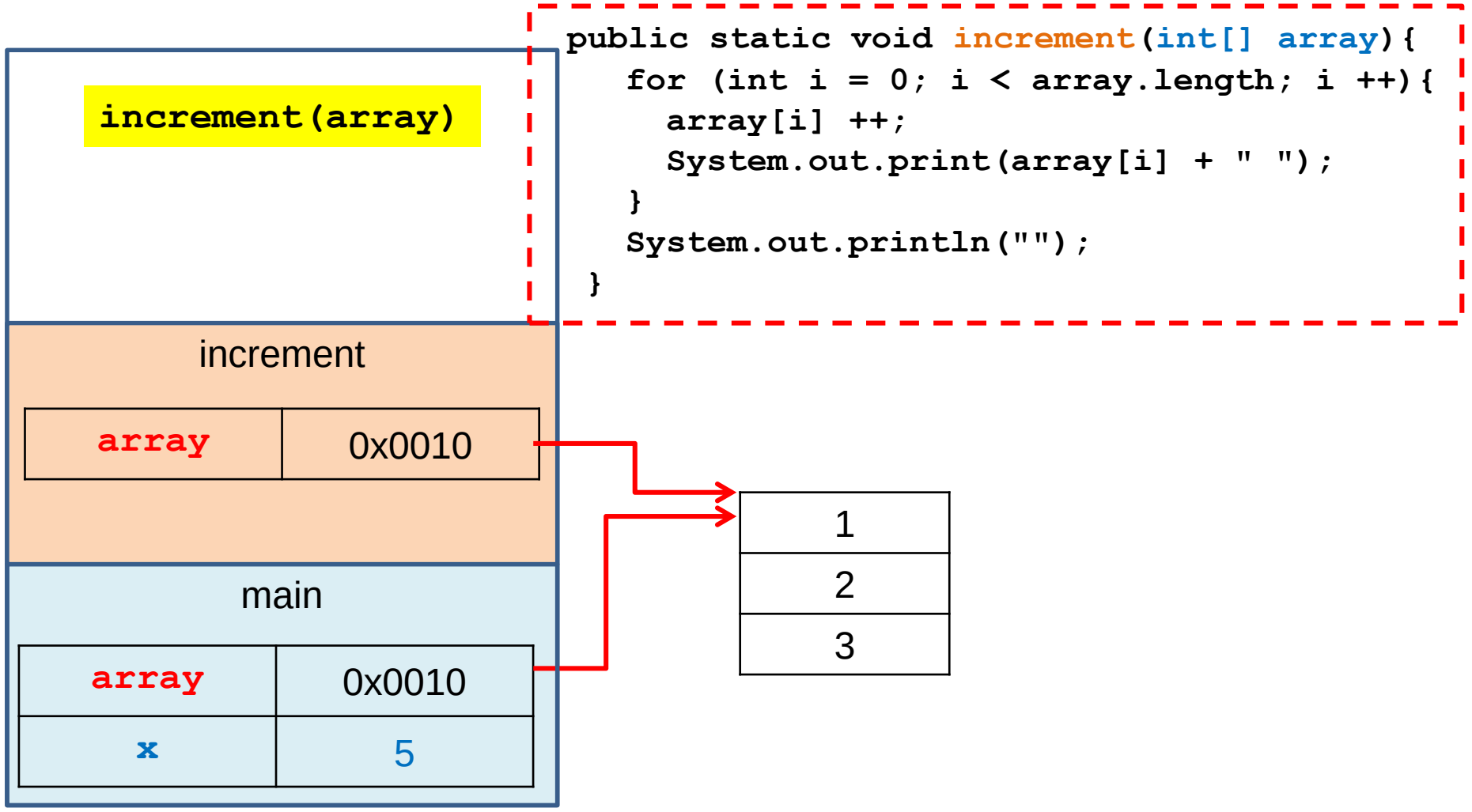
```
class ArrayVar
{
  public static void main(String[] args){
    int[] array = {1,2,3};
    int x = 5;

    increment(array);
    for (int i = 0; i < array.length; i ++){
      System.out.print(array[i] + " ");
    }
    System.out.println("");

    increment(x);
    System.out.println("x: " + x);

  }

  public static void increment(int[] array){
    for (int i = 0; i < array.length; i ++){
      array[i] ++;
      System.out.print(array[i] + " ");
    }
    System.out.println("");
  }

  public static void increment(int x)
  {
    x ++ ;
    System.out.println("x: " + x);
  }
}
```

Τι θα τυπώσει?

Πέρασμα παραμέτρων

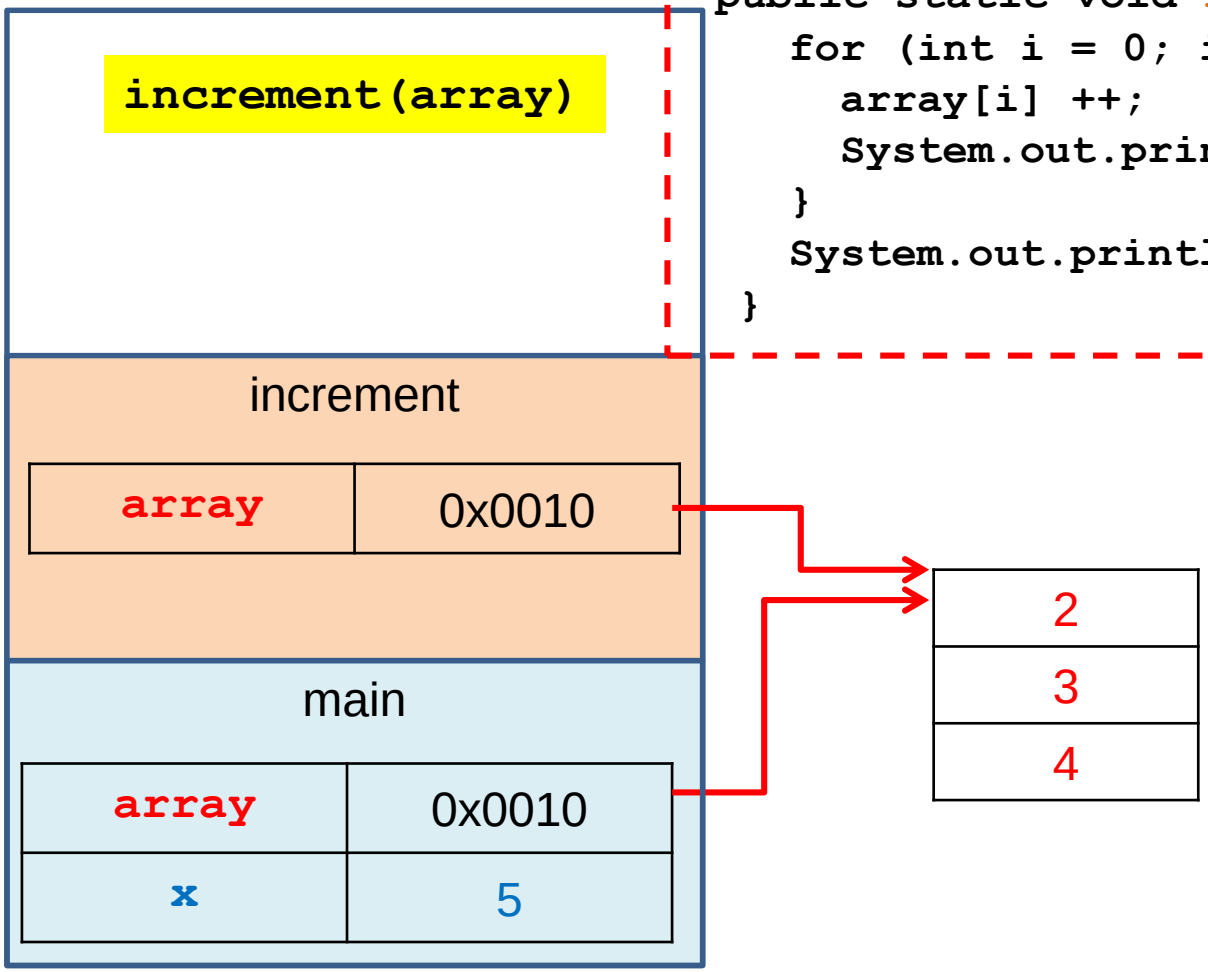


Πέρασμα παραμέτρων

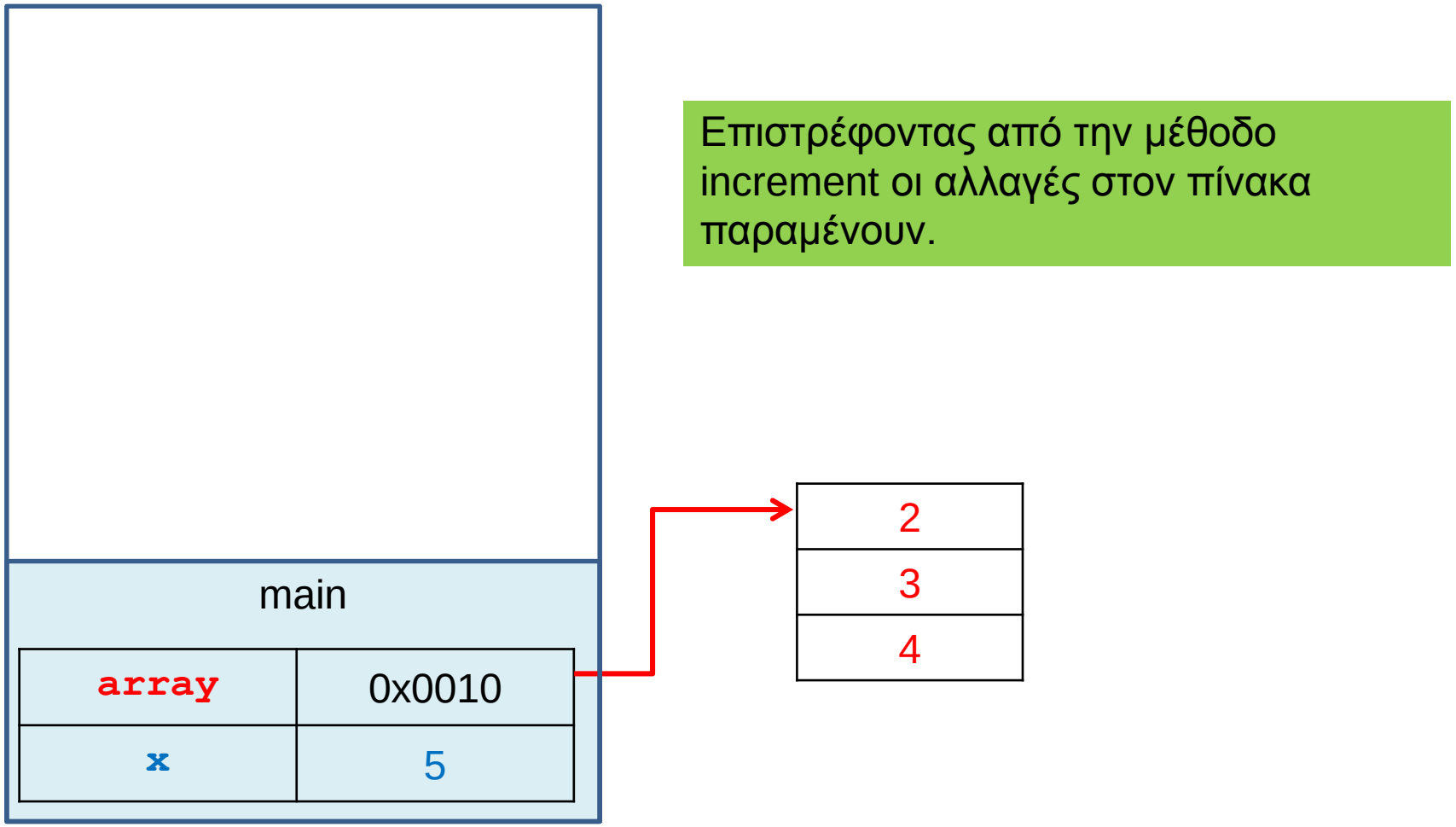


Πέρασμα παραμέτρων

```
public static void increment(int[] array){
   for (int i = 0; i < array.length; i ++){
     array[i] ++;
     System.out.print(array[i] + " ");
   }
   System.out.println("");
}
```

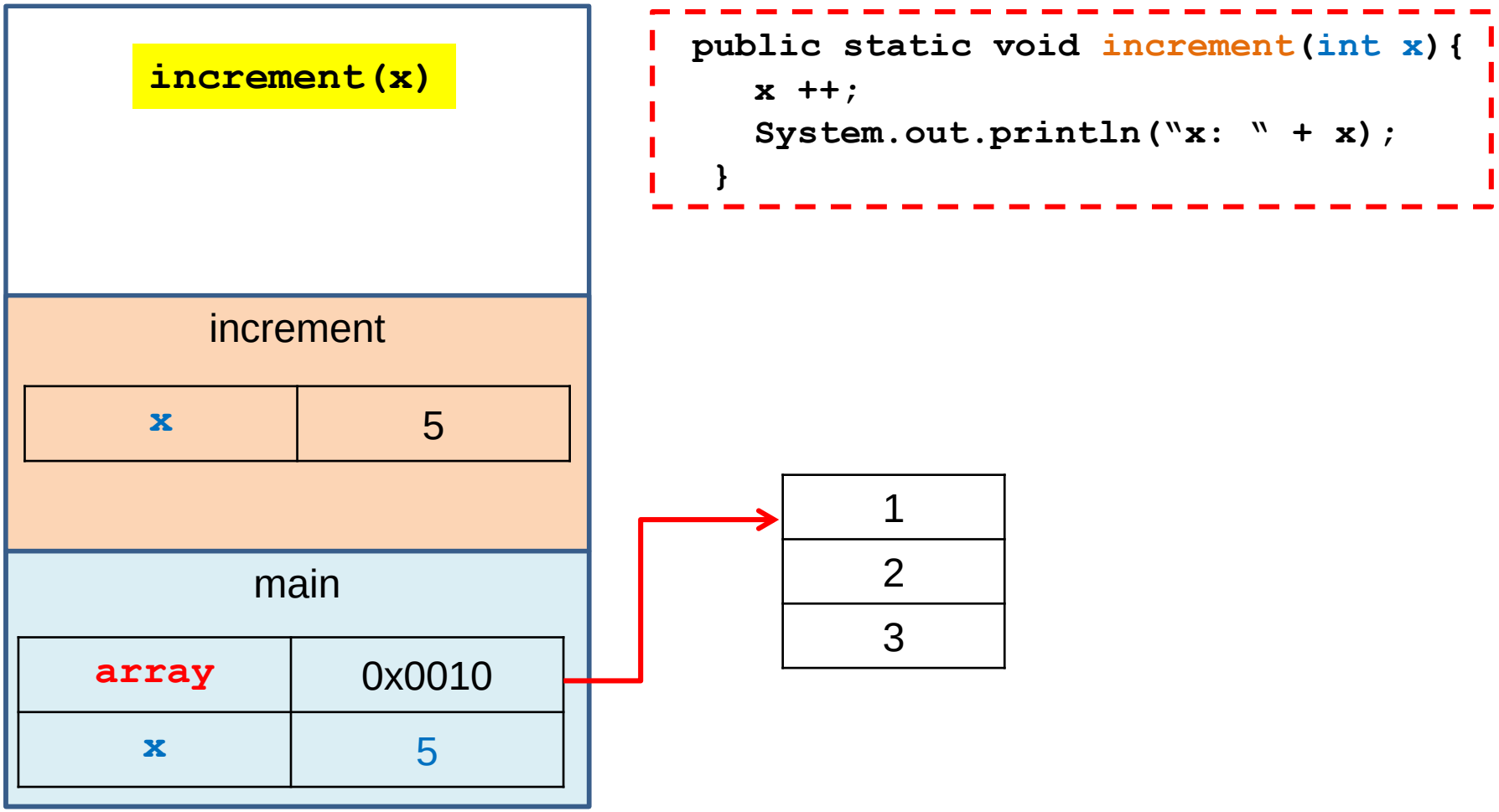


Πέρασμα παραμέτρων



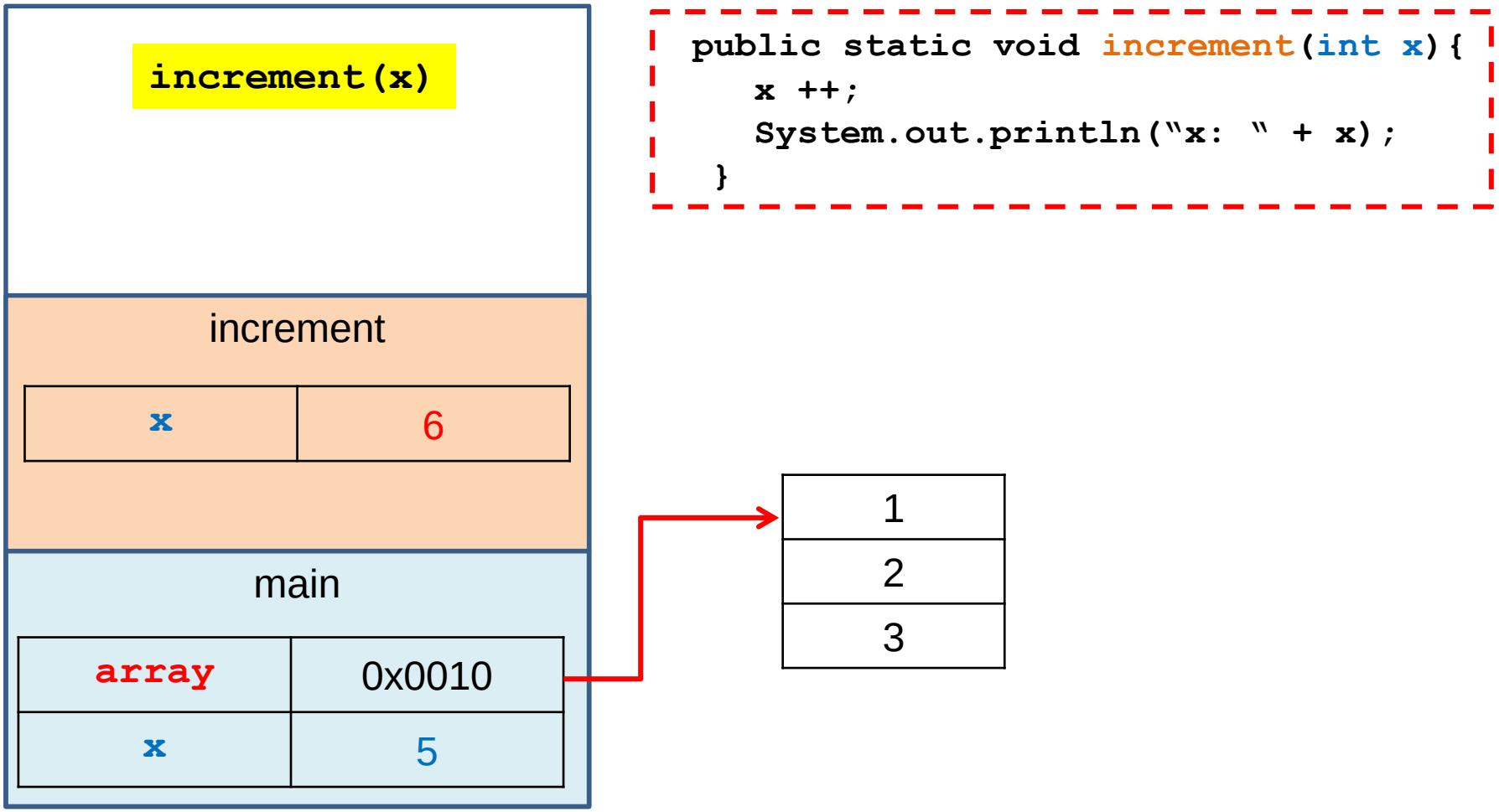
Επιστρέφοντας από την μέθοδο increment οι αλλαγές στον πίνακα παραμένουν.

Πέρασμα παραμέτρων



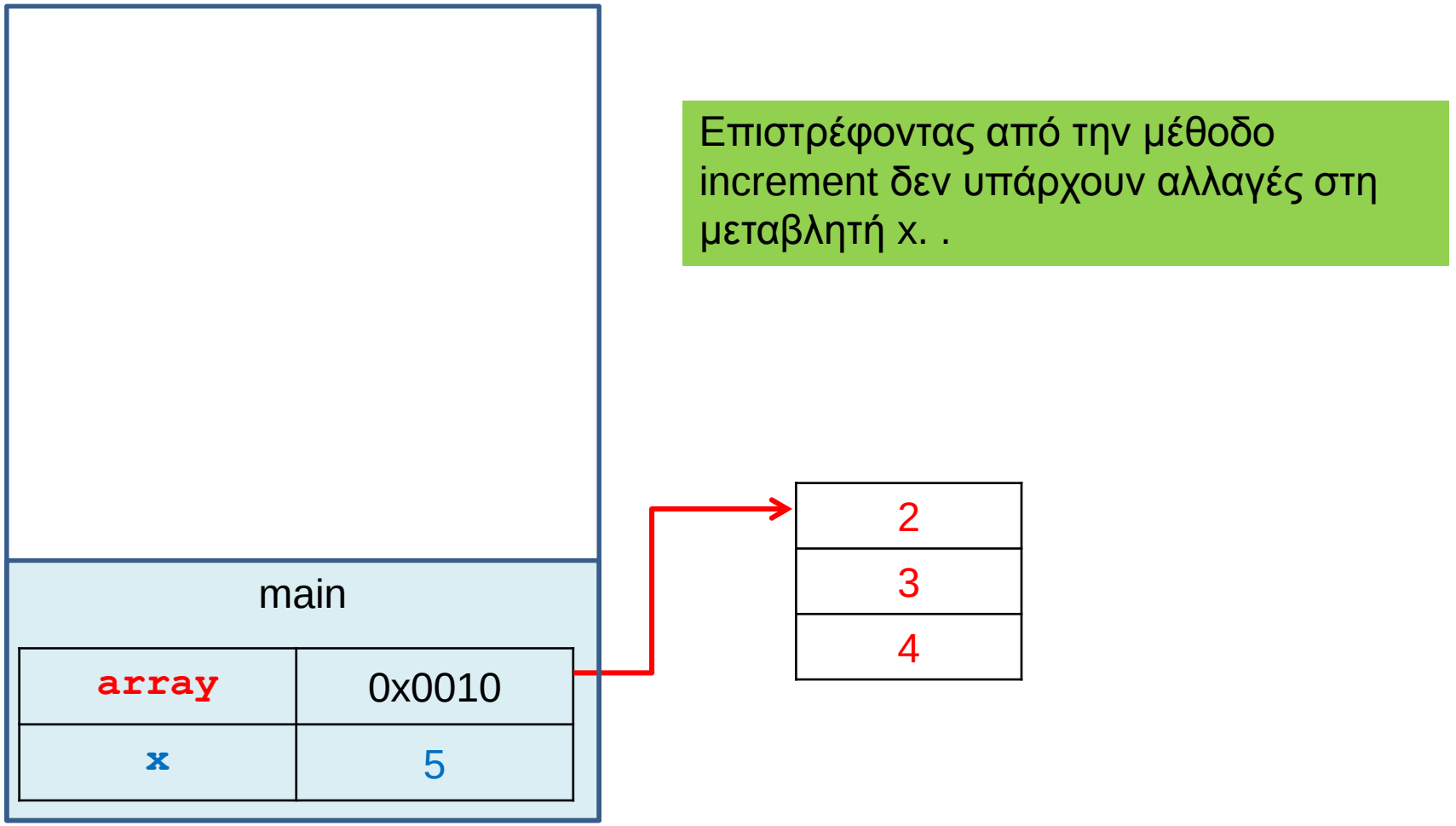

```
public static void increment(int x){
    x ++;
    System.out.println("x: " + x);
 }
```

Πέρασμα παραμέτρων




```
public static void increment(int x){
    x ++;
    System.out.println("x: " + x);
 }
```

Πέρασμα παραμέτρων



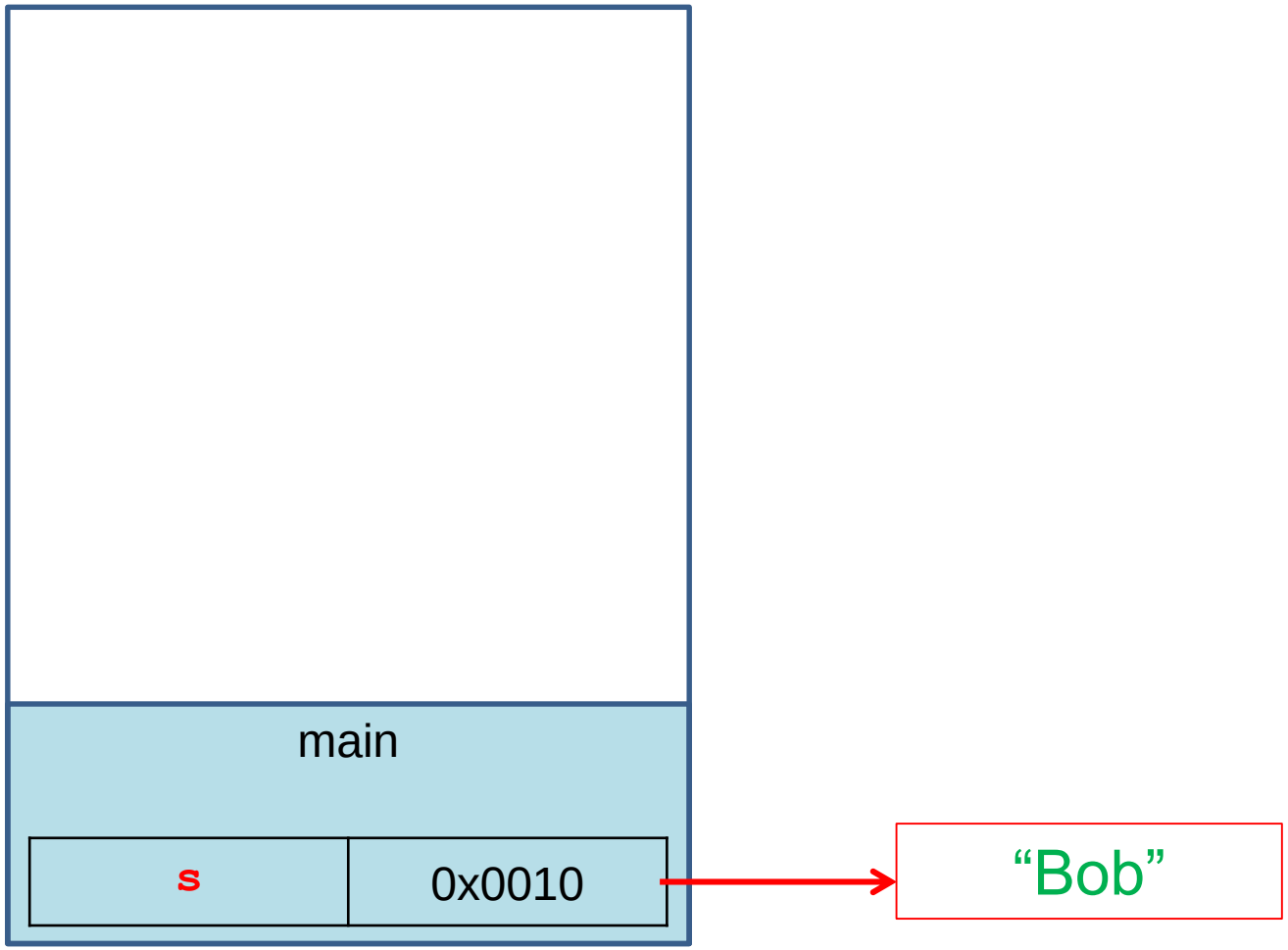
Επιστρέφοντας από την μέθοδο increment δεν υπάρχουν αλλαγές στη μεταβλητή x. .

Άλλο ένα παράδειγμα

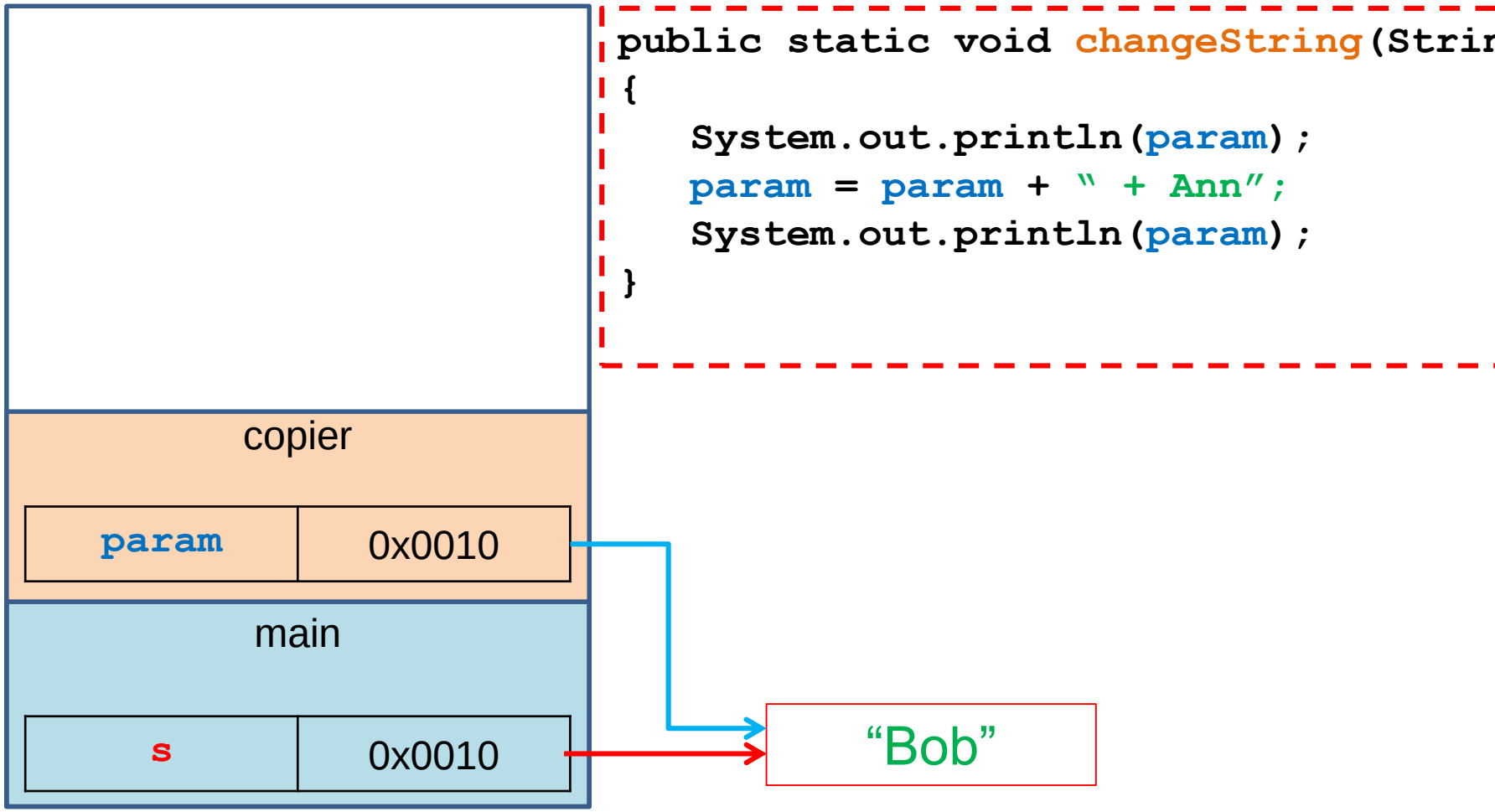
```
public class StringParameterDemo
{
    public static void main(String[] args)
    {
        String s = "Bob";
        changeString(s);
        System.out.println(s);
     }

    public static void changeString(String param)
    {
        System.out.println(param);
        param = param + " + Ann";
        System.out.println(param);
    }
}
```

Τι θα τυπώσει?

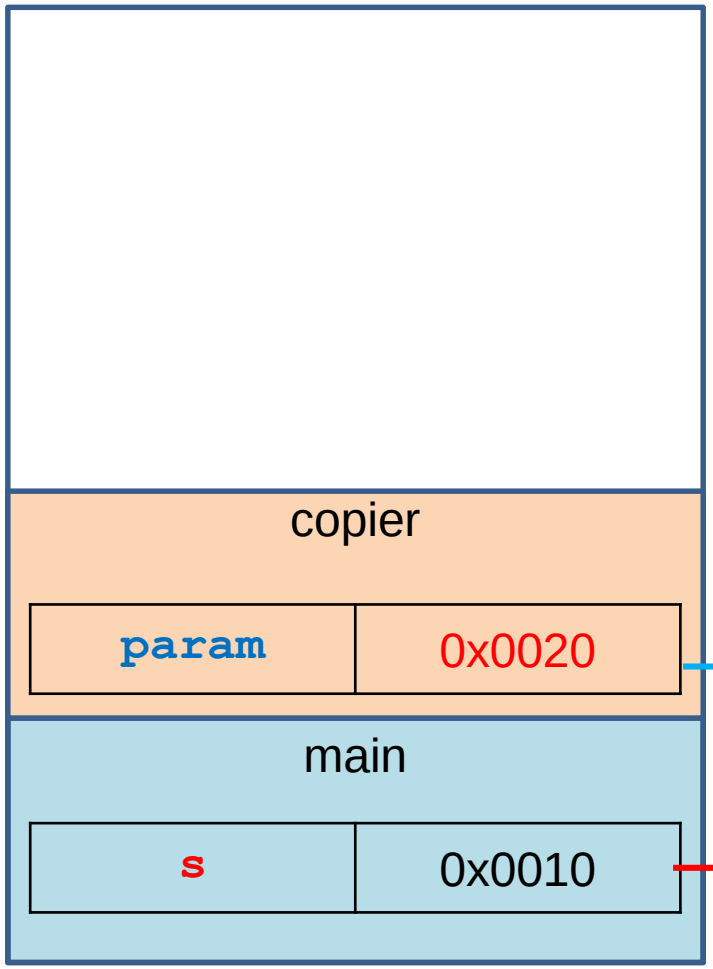
Εξέλιξη του προγράμματος



Εξέλιξη του προγράμματος



Εξέλιξη του προγράμματος




```
public static void changeString(Strir
{
    System.out.println(param);
    param = param + " + Ann";
    System.out.println(param);
}
```

Τα Strings είναι μη μεταλλάξιμα αντικείμενα (immutable objects)

"Bob + Ann"

"Bob"

Η αλλαγή σε ένα String έχει ως αποτέλεσμα τη δημιουργία ενός καινούριου αντικειμένου

String Interning

- Στην Java για κάθε string value που εμφανίζεται δημιουργείται ένα αντικείμενο, το οποίο ονομάζεται intern string, και το οποίο κρατάει αυτή την τιμή.
- Για αυτό και οι αλφαριθμητικές σταθερές μπορούν να χρησιμοποιηθούν και σαν αντικείμενα. Π.χ. μπορούμε να καλέσουμε:

```
"java".length()
```

- Αυτό μπορεί να προκαλέσει μπερδέματα με ελέγχους ισότητας.

Ισότητα String

Τι θα εκτυπωθεί?

```
import java.util.Scanner;

class StringEquality{
  public static void main(String[] args){
    Scanner input = new Scanner(System.in);

    String x = input.next();
    String z = new String("java");
    String y = "java";

    System.out.println("1. "+ (x == "java"));
    System.out.println("2. "+ (y == "java"));
    System.out.println("3. "+ (z == "java"));
    System.out.println("4. "+ x.equals("java"));
    System.out.println("5. "+ y.equals("java"));
    System.out.println("6. "+ z.equals("java"));
  }
}
```

1. false
2. true
3. false
4. true
5. true
6. true

Για την σύγκριση Strings **ΠΑΝΤΑ** χρησιμοποιούμε την μέθοδο equals.

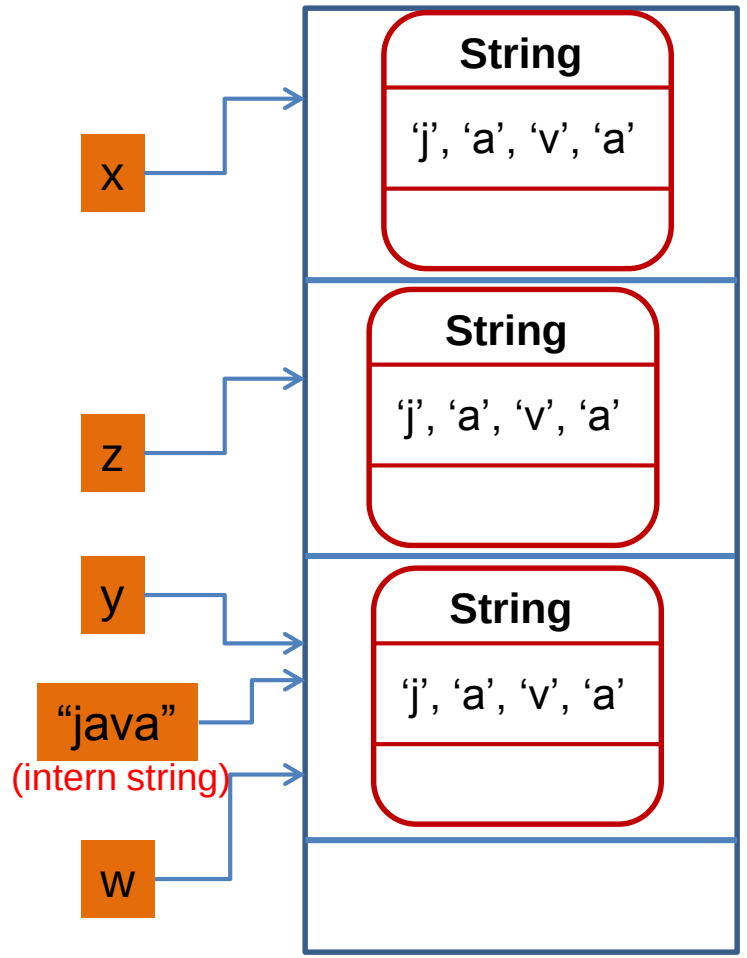
String Interning

```
String x = input.next();
String z = new String("java");
String y = "java";
String w = “java”;
```

- Όταν γίνεται η εκχώρηση της τιμής `"java"` δημιουργείται ένα intern string, και το οποίο κρατάει αυτή την τιμή.
- Η εντολή

```
String y = "java";
```

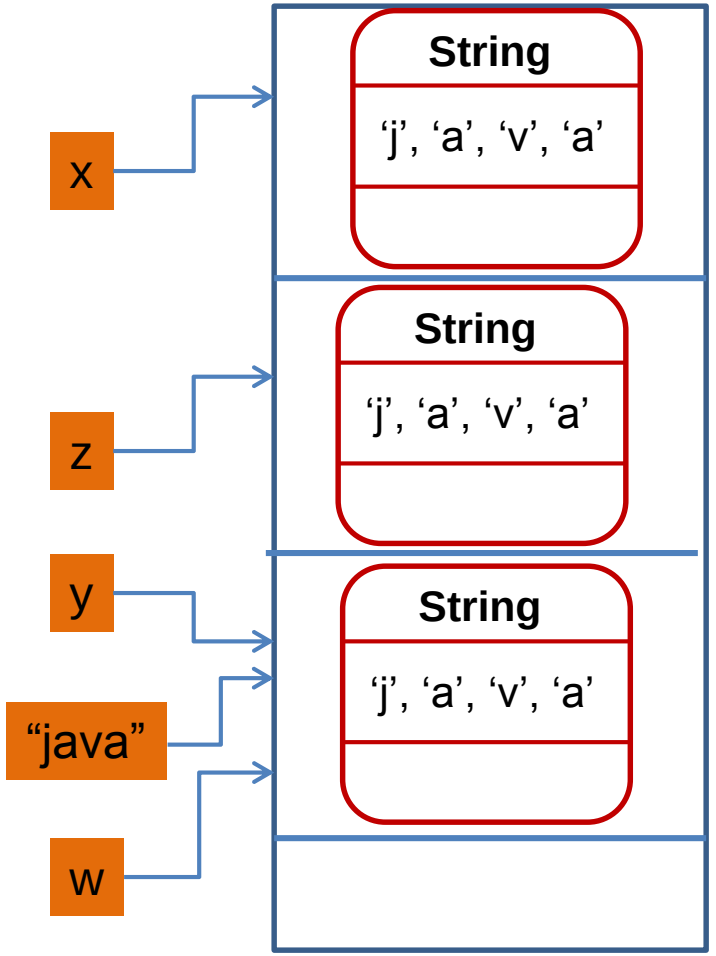
κάνει το `y` να δείχνει στη θέση που είναι αποθηκευμένη η τιμή `“java”`



String Interning

```
String x = input.next();
String z = new String("java");
String y = "java";
String w = "java";
System.out.println((y == "java");
```

- Ο τελεστής == μεταξύ δύο αντικειμένων εξετάζει αν πρόκειται για την ίδια θέση μνήμης.
- Γι αυτό (`y == "java"`) επιστρέφει true.



Equals

- Έχουμε πει ότι όταν ελέγχουμε ισότητα μεταξύ αντικειμένων (π.χ., Strings) πρέπει να γίνεται μέσω της μεθόδου equals και όχι με το ==
- Η συζήτηση με τις αναφορές εξηγεί γιατί η σύγκριση με == δε δουλεύει

- Η σύγκριση με == συγκρίνει αν δύο αναφορές είναι ίδιες και όχι αν τα περιεχόμενα των θέσεων μνήμης στις οποίες δείχνουν οι αναφορές είναι ίδια.

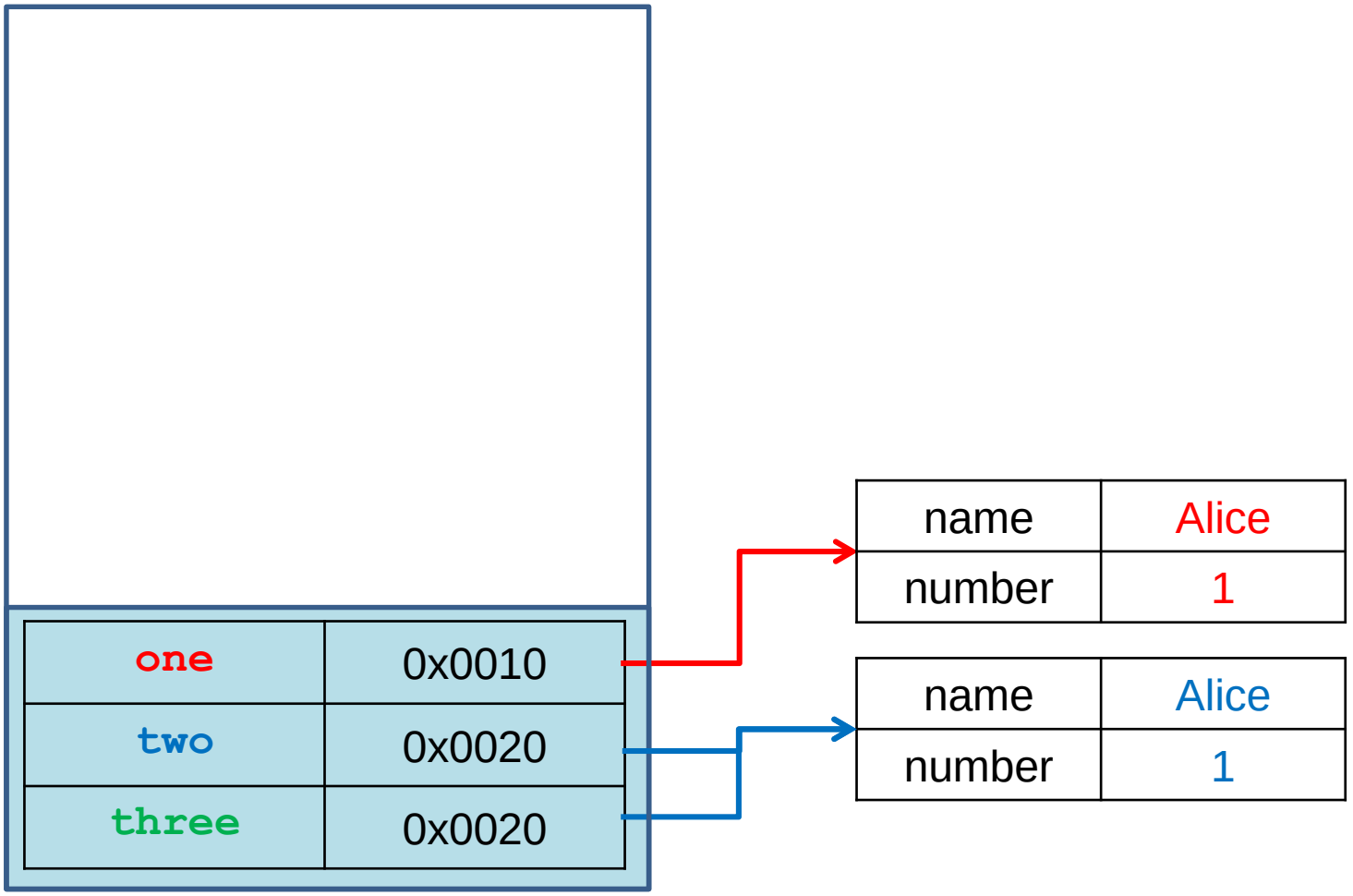
Παράδειγμα

- Τι θα τυπώσει ο παρακάτω κώδικας?

```
Person one = new Person("Alice", 1);
Person two = new Person("Alice",1);
Person three = two;
System.out.println(one == two);
System.out.println(two == three);
System.out.println(one == three);
System.out.println(one.equals(two));
System.out.println(two.equals(three));
System.out.println(one.equals(three));
```

false
true
false
true
true
true

Εξήγηση


one
0x0010
two
0x0020
three
0x0020
name
Alice
number
1
name
Alice
number
1

```
class StringClass
{
    String s = "abc";

    public void changeObject(ClassWithStrings other){
        if (this.s == other.s){
            System.out.println("Same");
        }else {
            System.out.println("Different");
        }
        String local = new String("local");
        other.s = local;
        local = "local";
        s = local;
        if (this.s == other.s){
            System.out.println("Same");
        }else {
            System.out.println("Different");
        }
    }

}
```

Η ανάθεση String σταθεράς έχει αποτέλεσμα την δημιουργία ενός intern string στο οποίο δείχνουν όλα τα strings στα οποία ανατίθεται η σταθερά.

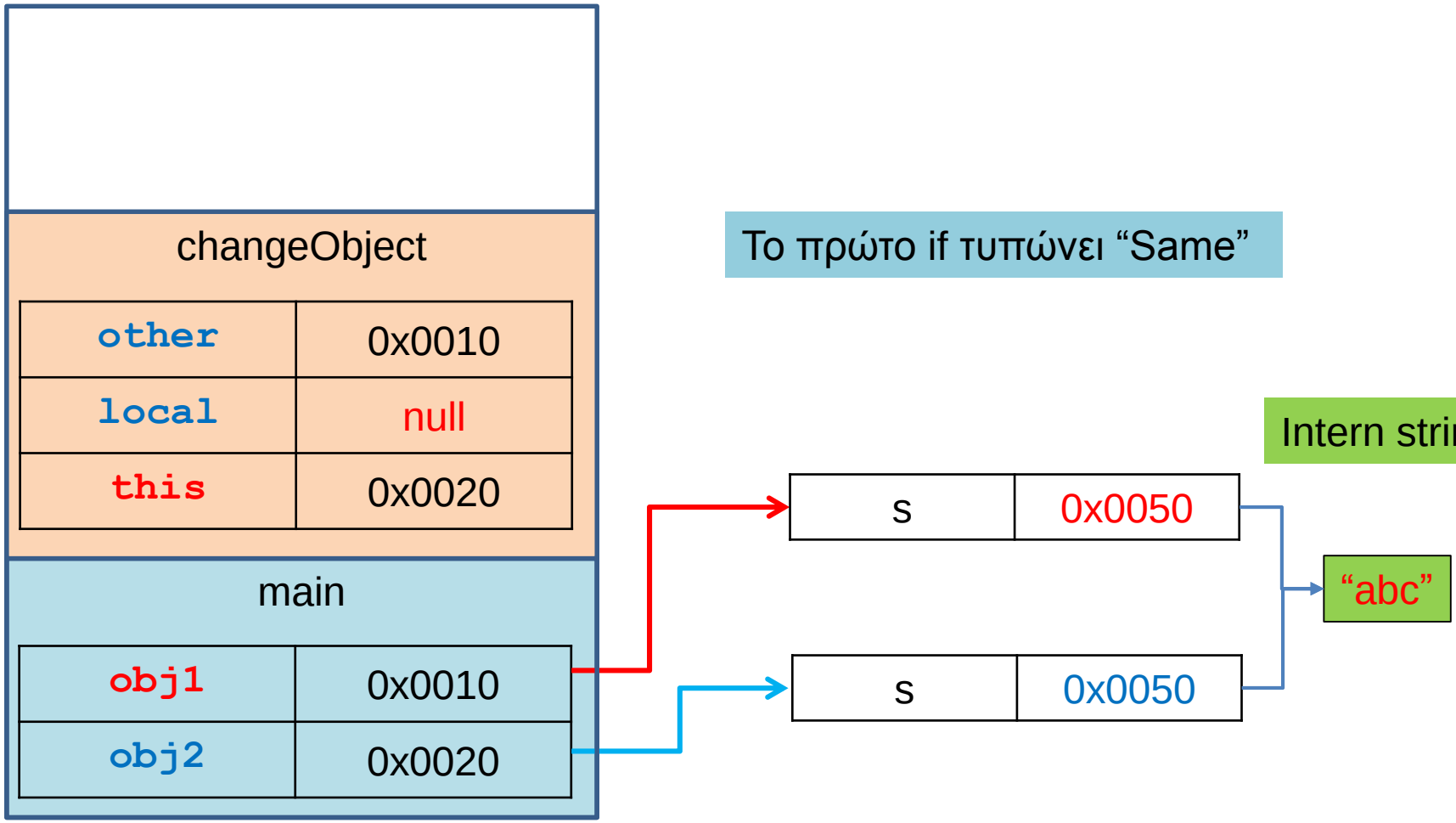
Η ανάθεση String σταθεράς είναι διαφορετική από τη δημιουργία αντικειμένου με new

Η σταθερά δημιουργεί ένα νέο intern String

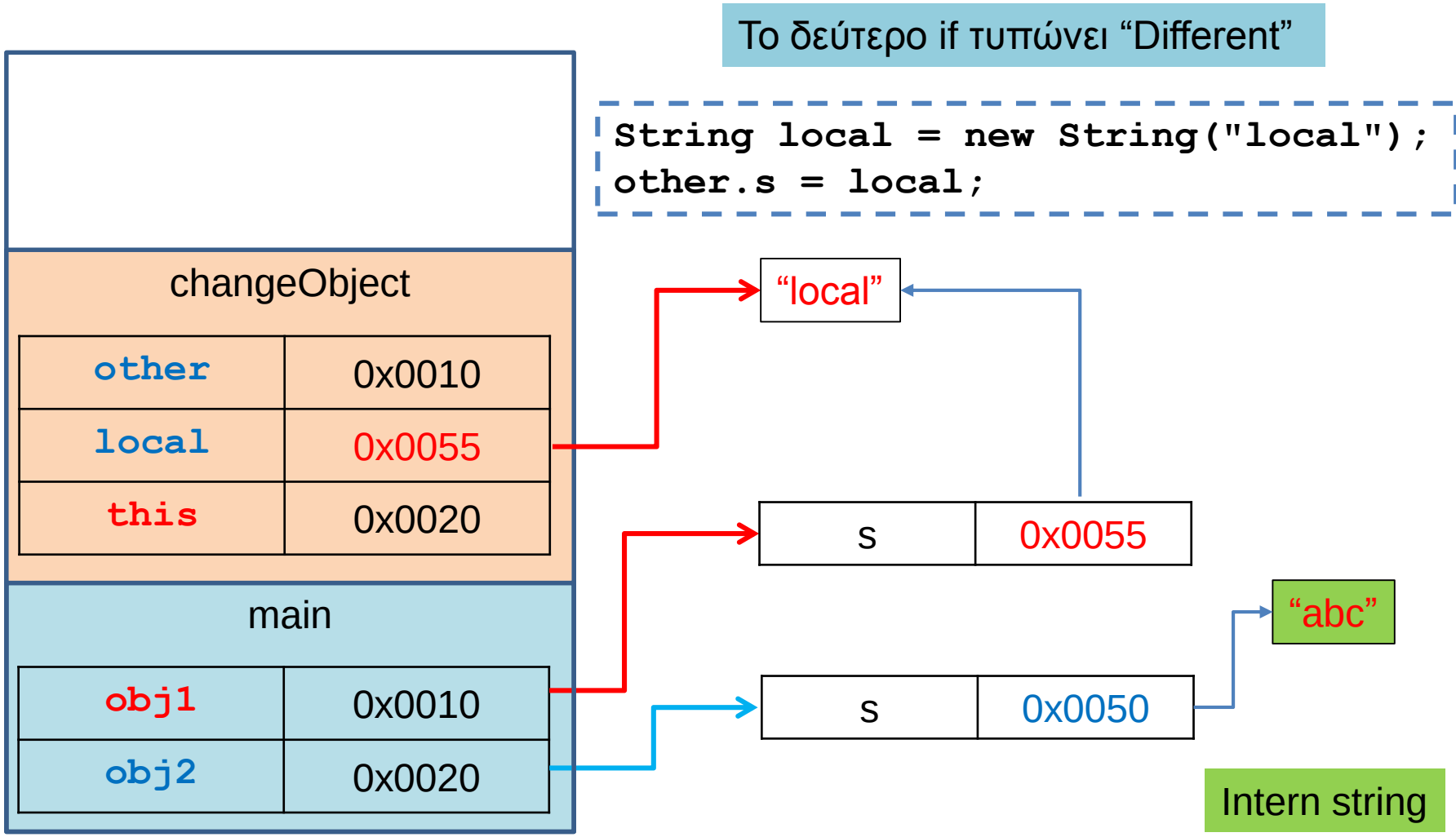
```
class StringTest2{
    public static void main(String[] args){
        StringClass obj1 = new StringClass();
        StringClass obj2 = new StringClass();
        obj2.changeObject(obj1);
    }
}
```

Τι θα τυπώσει?

Εξέλιξη του προγράμματος



Εξέλιξη του προγράμματος



Εξέλιξη του προγράμματος