ΤΕΧΝΙΚΕΣ ΑΝΤΙΚΕΙΜΕΝΟΣΤΡΑΦΟΥΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ

Αναφορές

Μέθοδοι που επιστρέφουν αντικείμενα

Deep and Shallow Copies

Μαθήματα από το εργαστήριο

- Όταν η εκφώνηση σας ζητάει να φτιάξετε μία μέθοδο που παίρνει σαν όρισμα μια κλάση που έχουμε ορίσει, αυτό σημαίνει ότι θέλουμε σαν όρισμα το αντικείμενο της κλάσης.
  - Στο εργαστήριο η μέθοδος deposit έπρεπε να πάρει σαν όρισμα μια επιταγή, δηλαδή ένα αντικείμενο της κλάσης Check.
  - ΌΧΙ το όνομα και το ποσό
- Για να διαβάσουμε τα πεδία του αντικειμένου χρησιμοποιούμε τις μεθόδους πρόσβασης (accessor methods)
  - Στο παράδειγμα μας τις μεθόδους getName και getAmount.

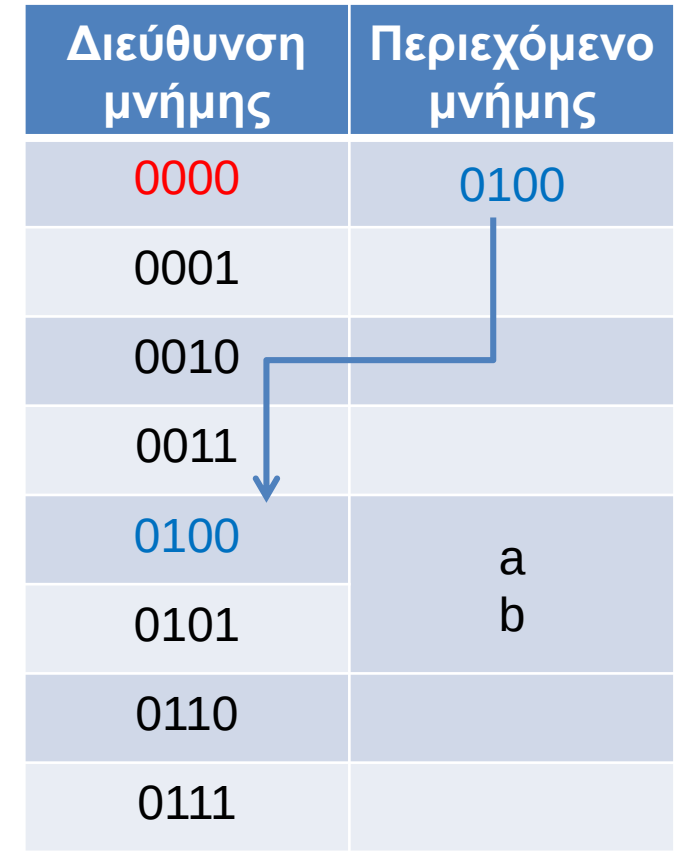
Η μέθοδος deposit

```
public boolean deposit(Check depositCheck){
      if (this.name.equals(depositCheck.getName())){
            this.amount += depositCheck.getAmount();
            return true;
      }
      return false;
}
```

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 = new int[3];
```

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

Παράδειγμα - πινάκες

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

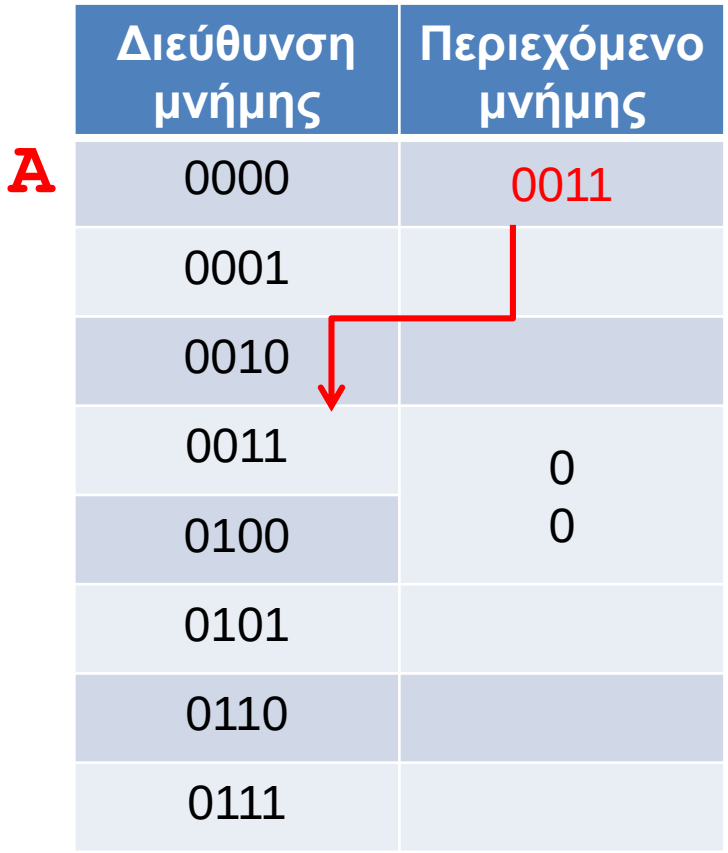
Η δεσμευμένη λέξη null σημαίνει μια κενή αναφορά (μια διεύθυνση που δεν δείχνει πουθενά)

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

Παράδειγμα - πινάκες

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

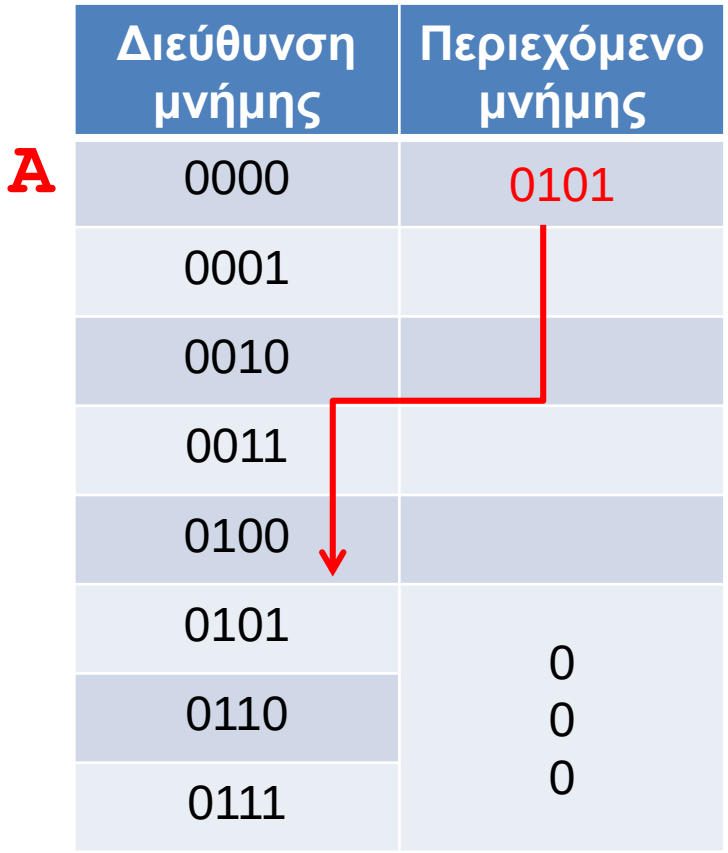
Με την εντολή new δεσμεύουμε δύο θέσεις ακεραίων και η αναφορά του A δείχνει σε αυτό το χώρο που δεσμεύσαμε



Παράδειγμα - πινάκες

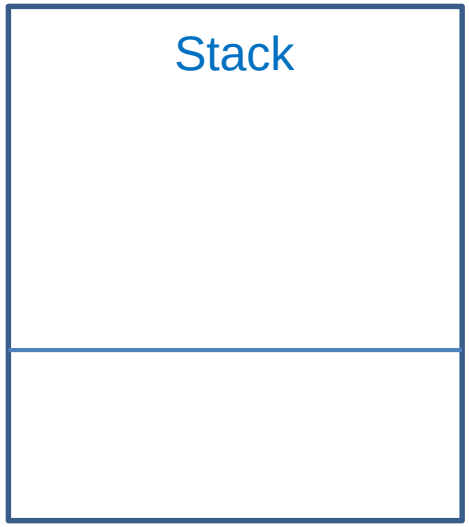
```
int[] A;
A = new int[2];
A = new int[3];
```

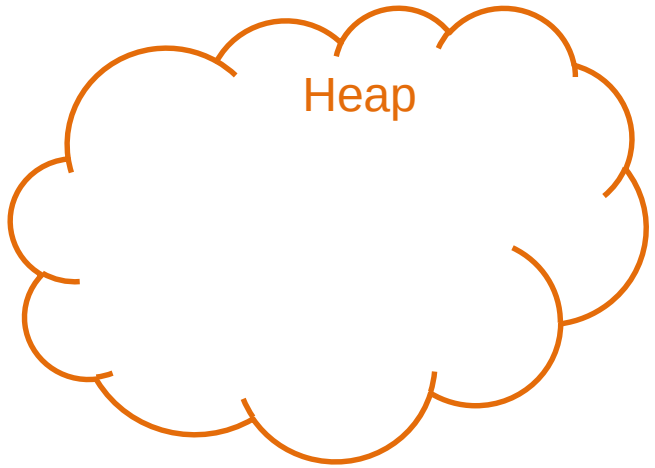
Με νέα κλήση της new δεσμεύουμε νέο χώρο για το Α, και αν δεν έχουμε κρατήσει την προηγούμενη αναφορά σε κάποια άλλη μεταβλητή τότε χάνεται (garbage collection)



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



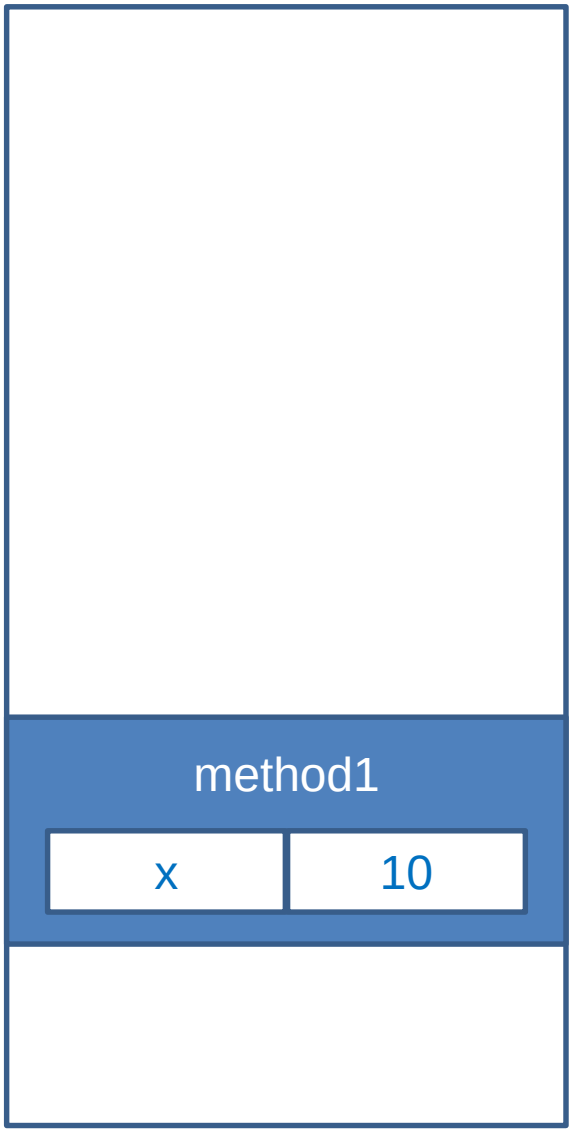


Stack

- 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.

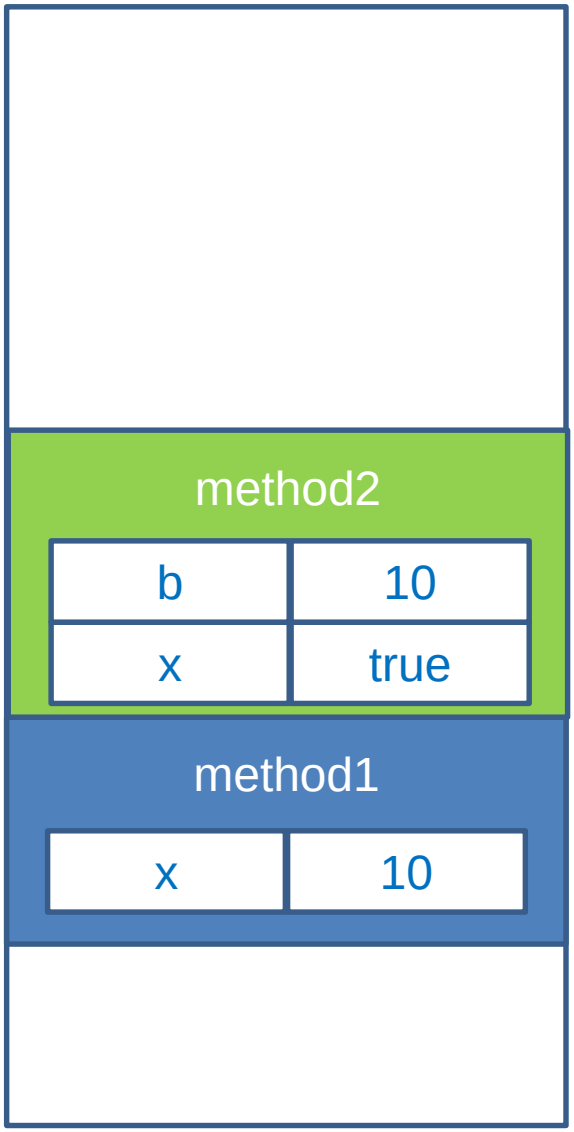
Παράδειγμα

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



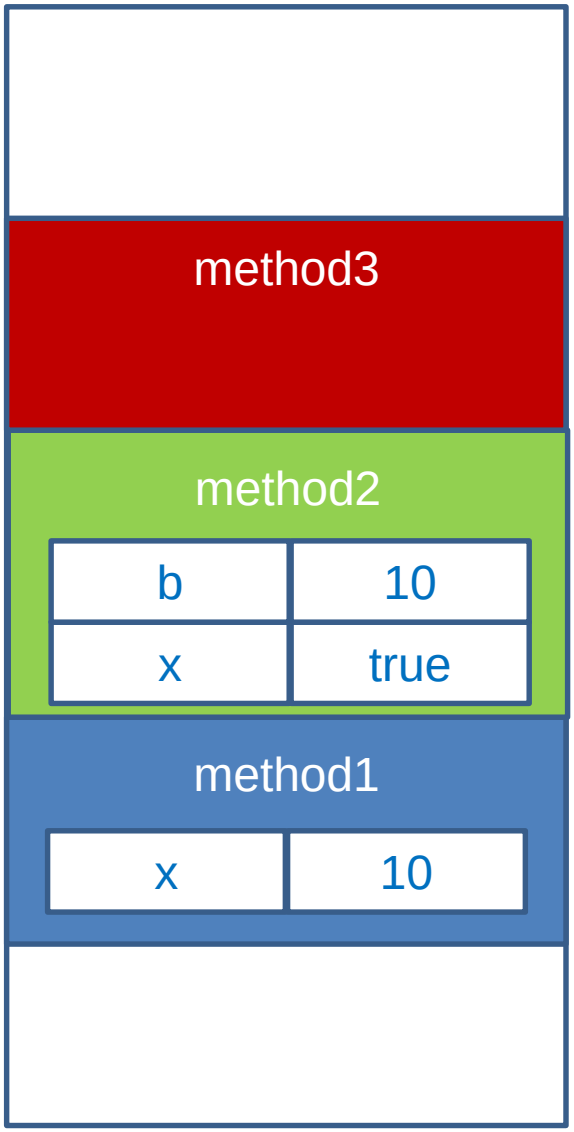
Παράδειγμα

```
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()
{…}
```



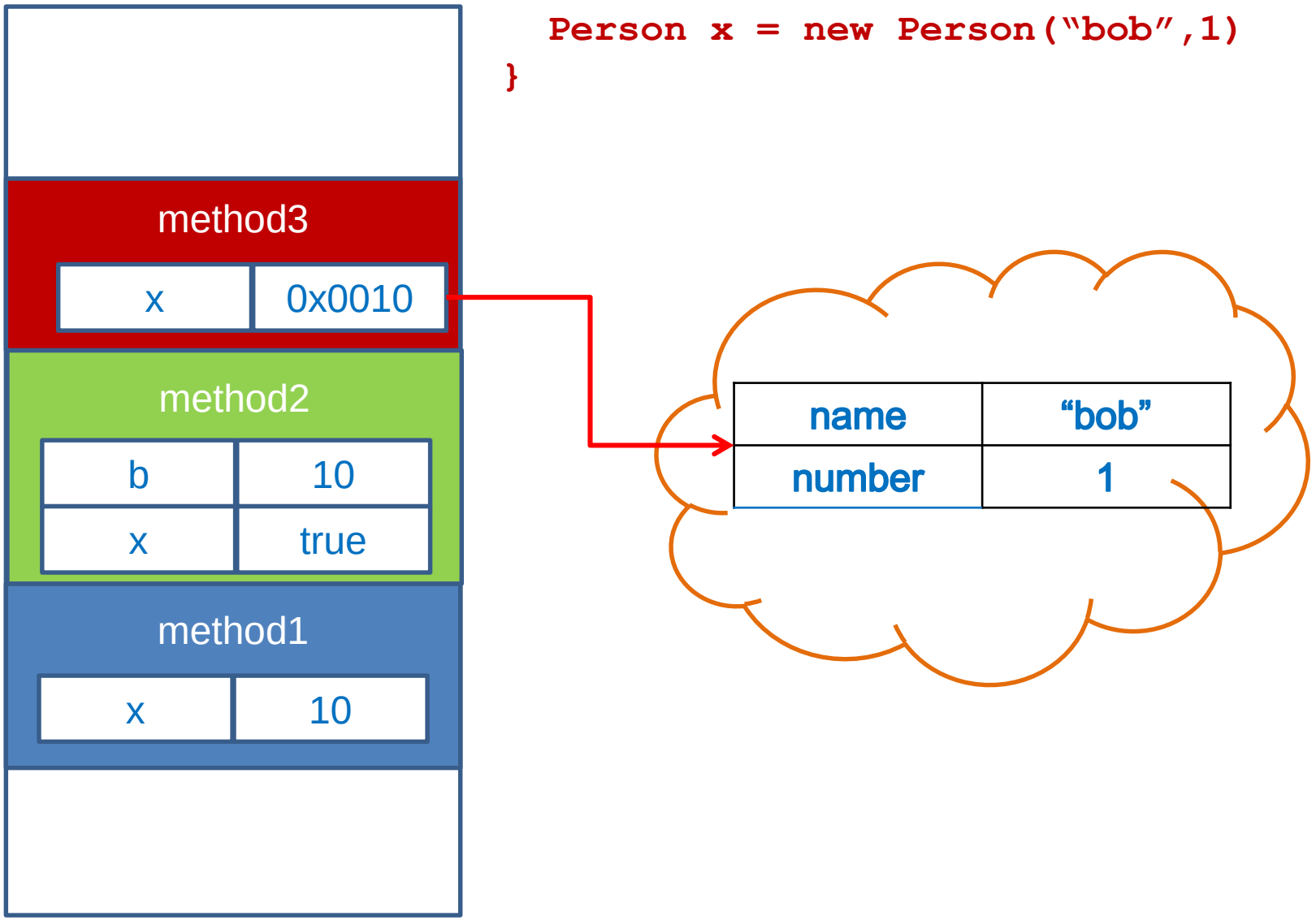
Heap

- 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.

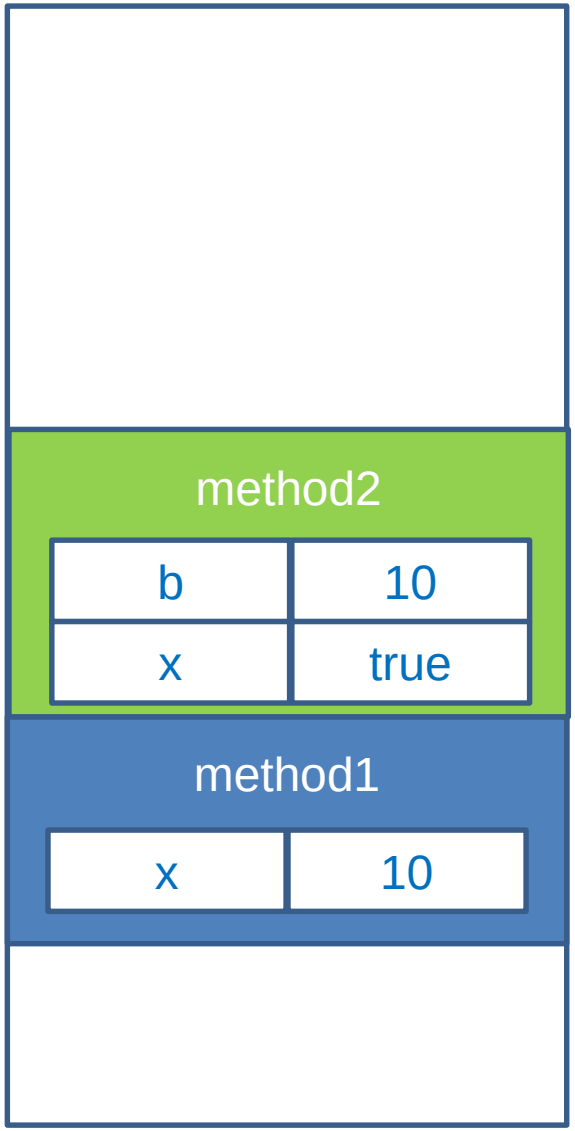
```
public class Person
{
    private String name;
    private int number;

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

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

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

Παράδειγμα

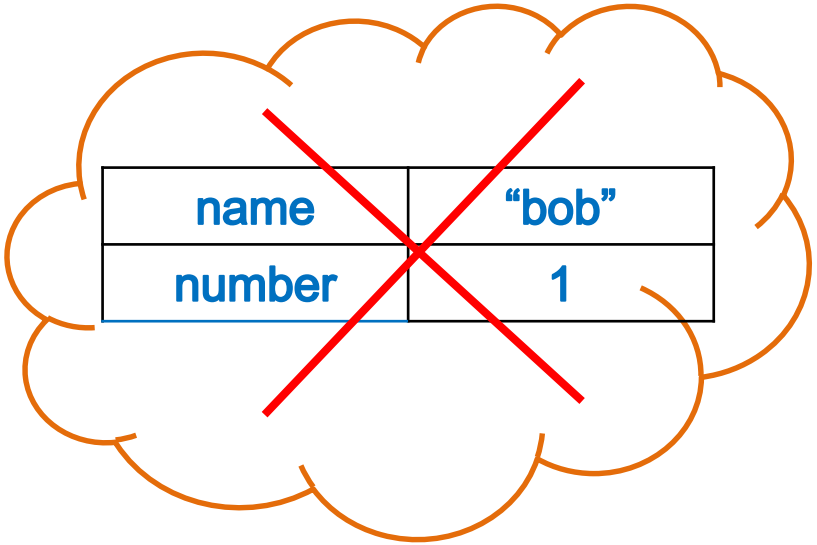
```
public void method3()
{
   Person x = new Person("bob",1)
}
```



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

Αντικείμενα ως παράμετροι

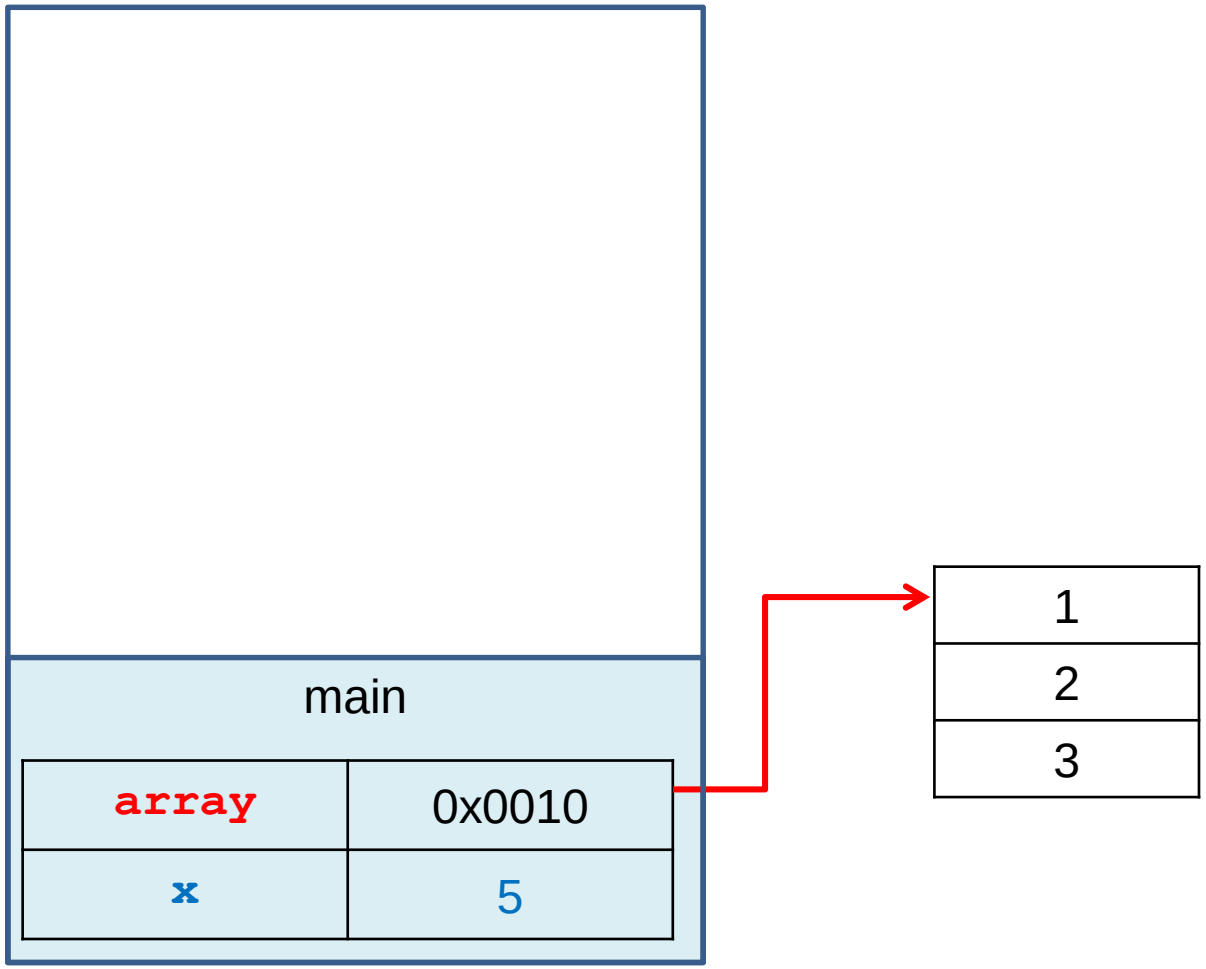
- Όταν περνάμε παραμέτρους σε μία μέθοδο το πέρασμα γίνεται πάντα δια τιμής (call-by-value)
  - Δηλαδή απλά περνάμε τα περιεχόμενα της θέσης μνήμης της συγκεκριμένης μεταβλητής.
  - Για μεταβλητές πρωταρχικού τύπου, αλλαγές στην τιμή της παραμέτρου δεν αλλάζουν την μεταβλητή που περάσαμε σαν όρισμα.

- 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.

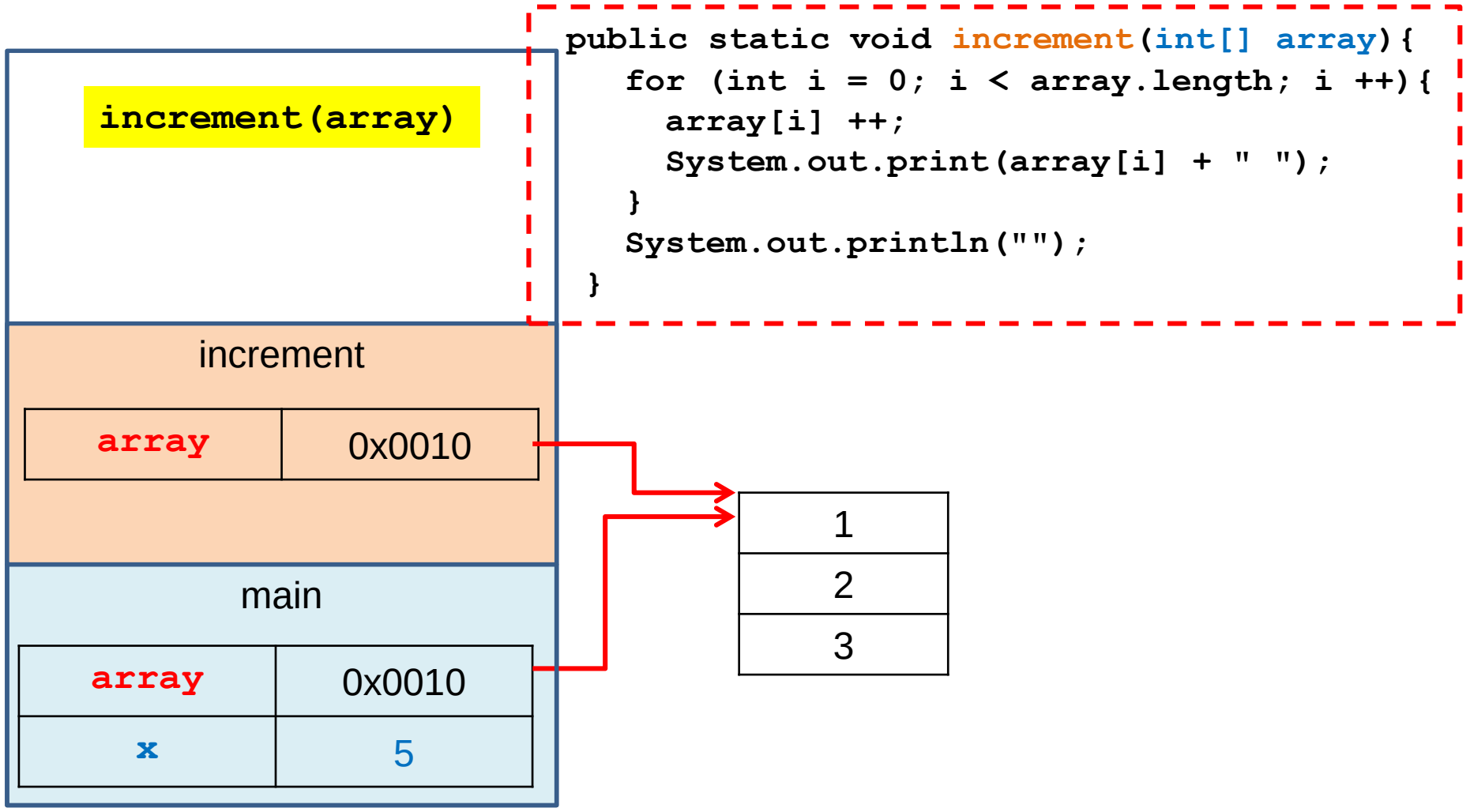
```
class ArrayVar
{
  public static void main(String[] args){
    int[] array = {1,2,3};
    int x = 5;

    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

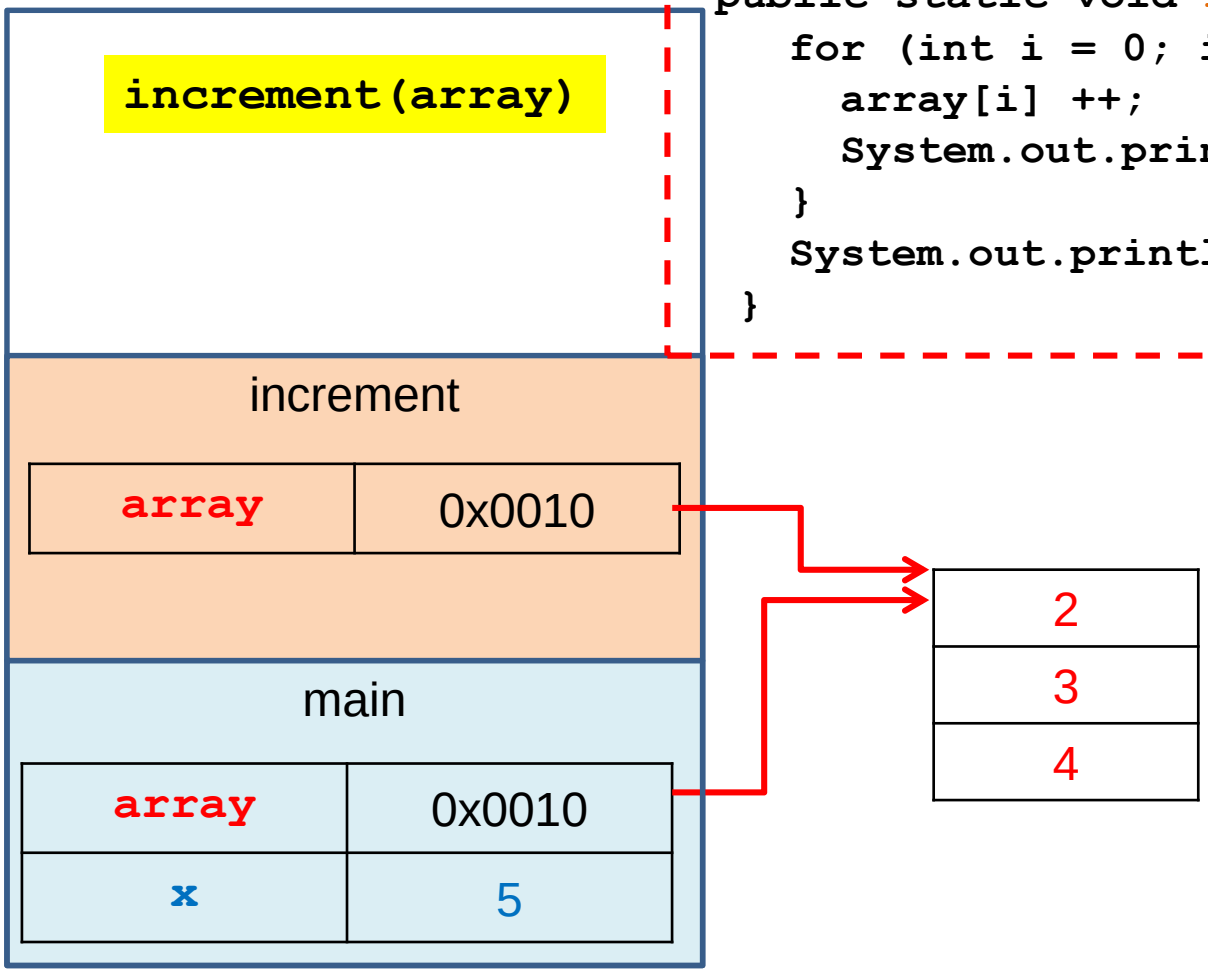


Πέρασμα παραμέτρων

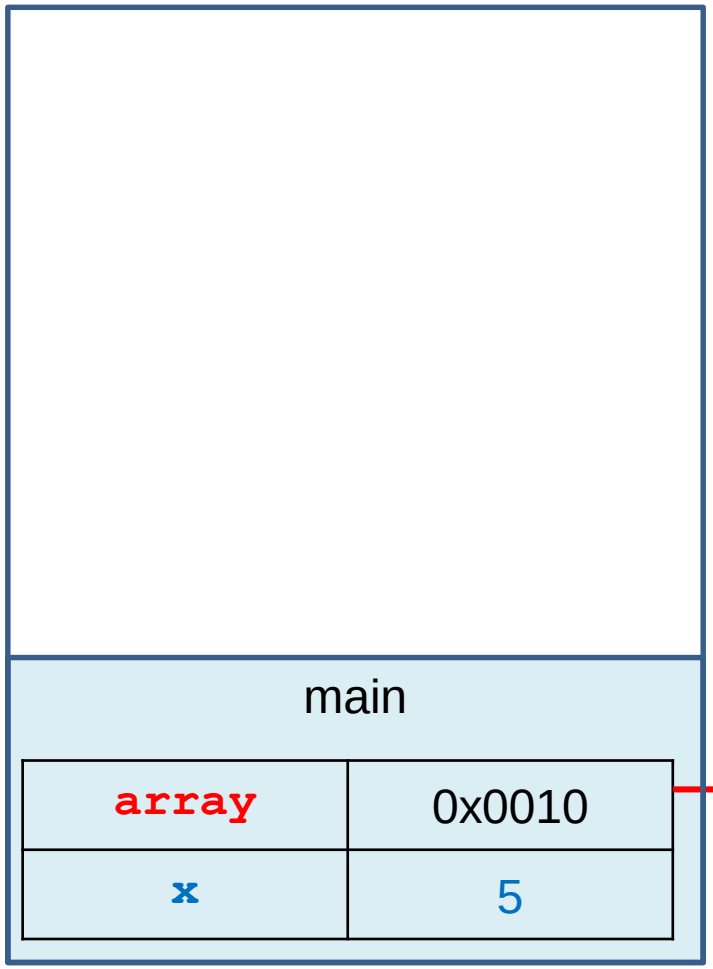


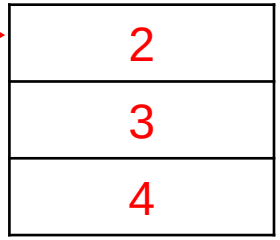
Πέρασμα παραμέτρων

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



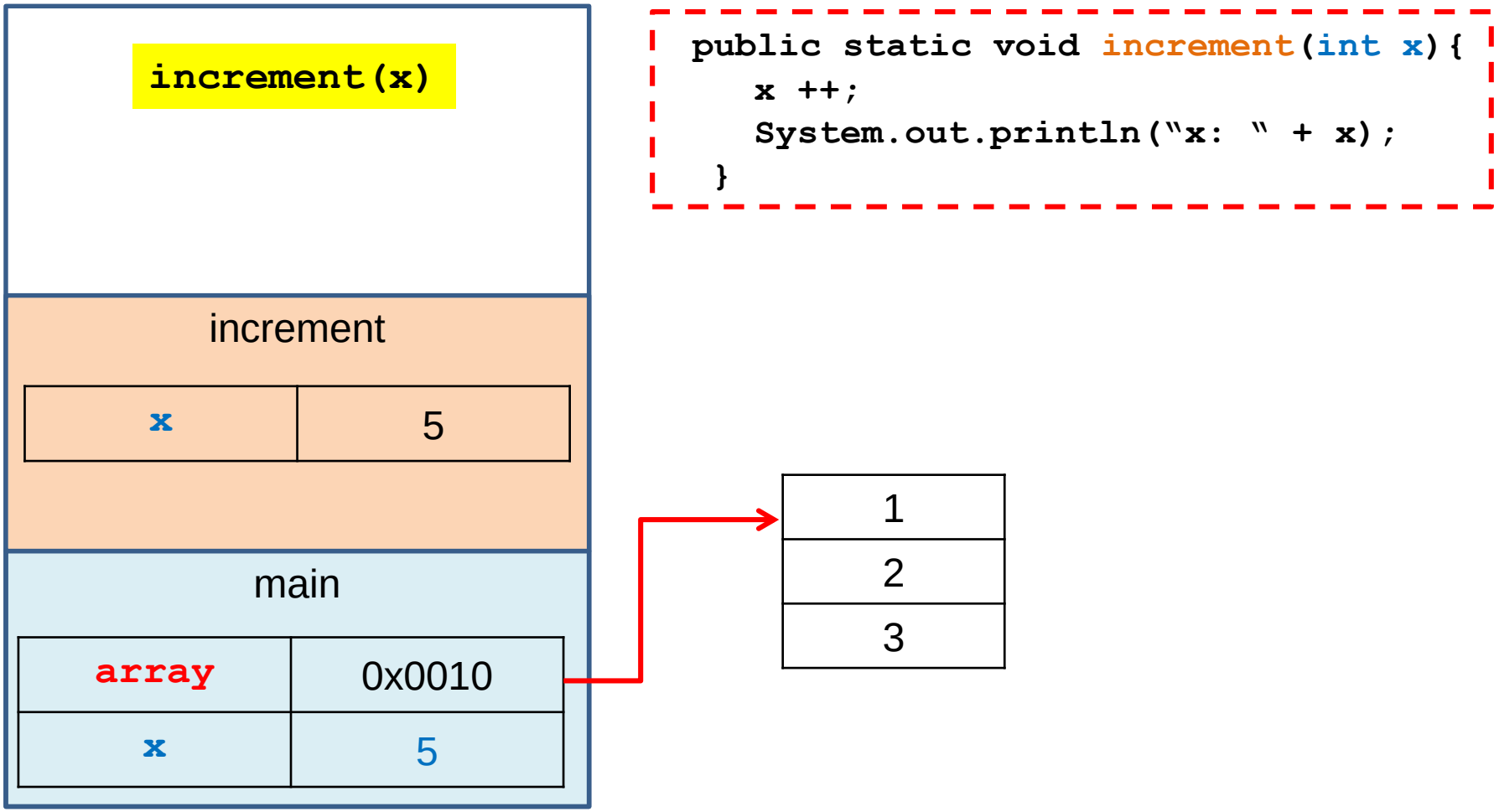
Πέρασμα παραμέτρων





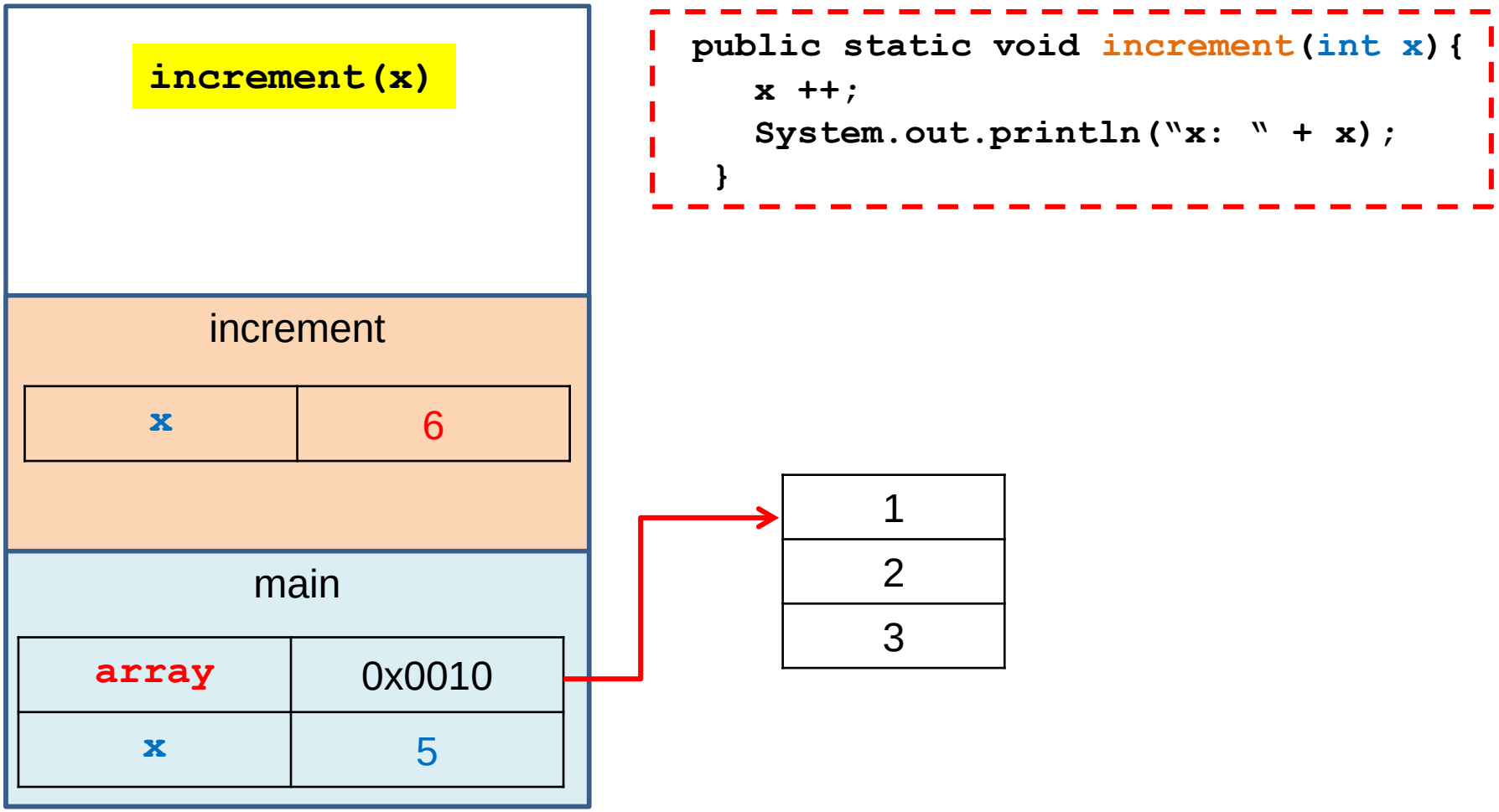
Επιστρέφοντας από την μέθοδο increment οι αλλαγές στον πίνακα παραμένουν.

Πέρασμα παραμέτρων



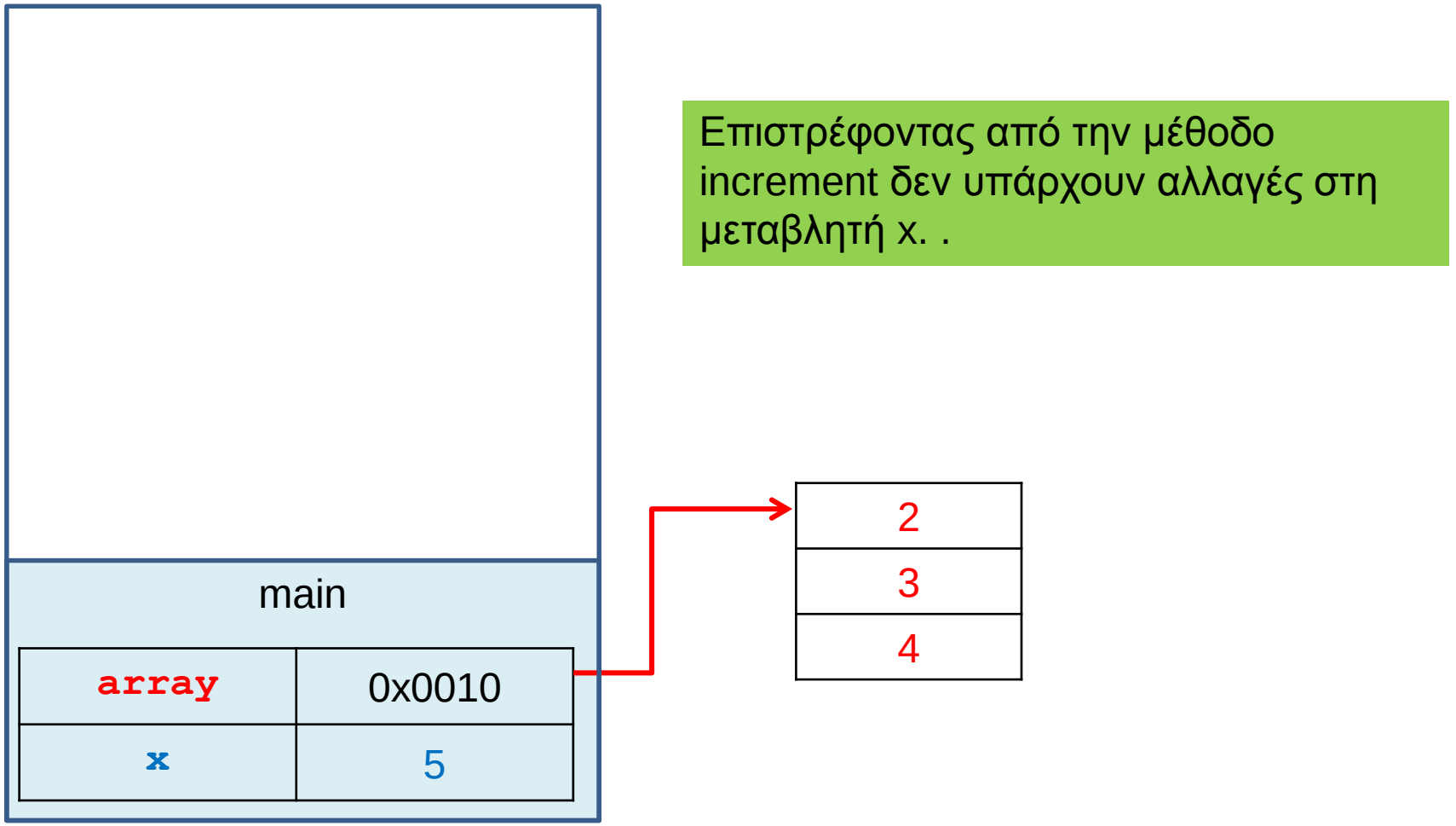

```
public static void increment(int x){
    x ++;
    System.out.println("x: " + x);
 }
```

Πέρασμα παραμέτρων




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

Πέρασμα παραμέτρων

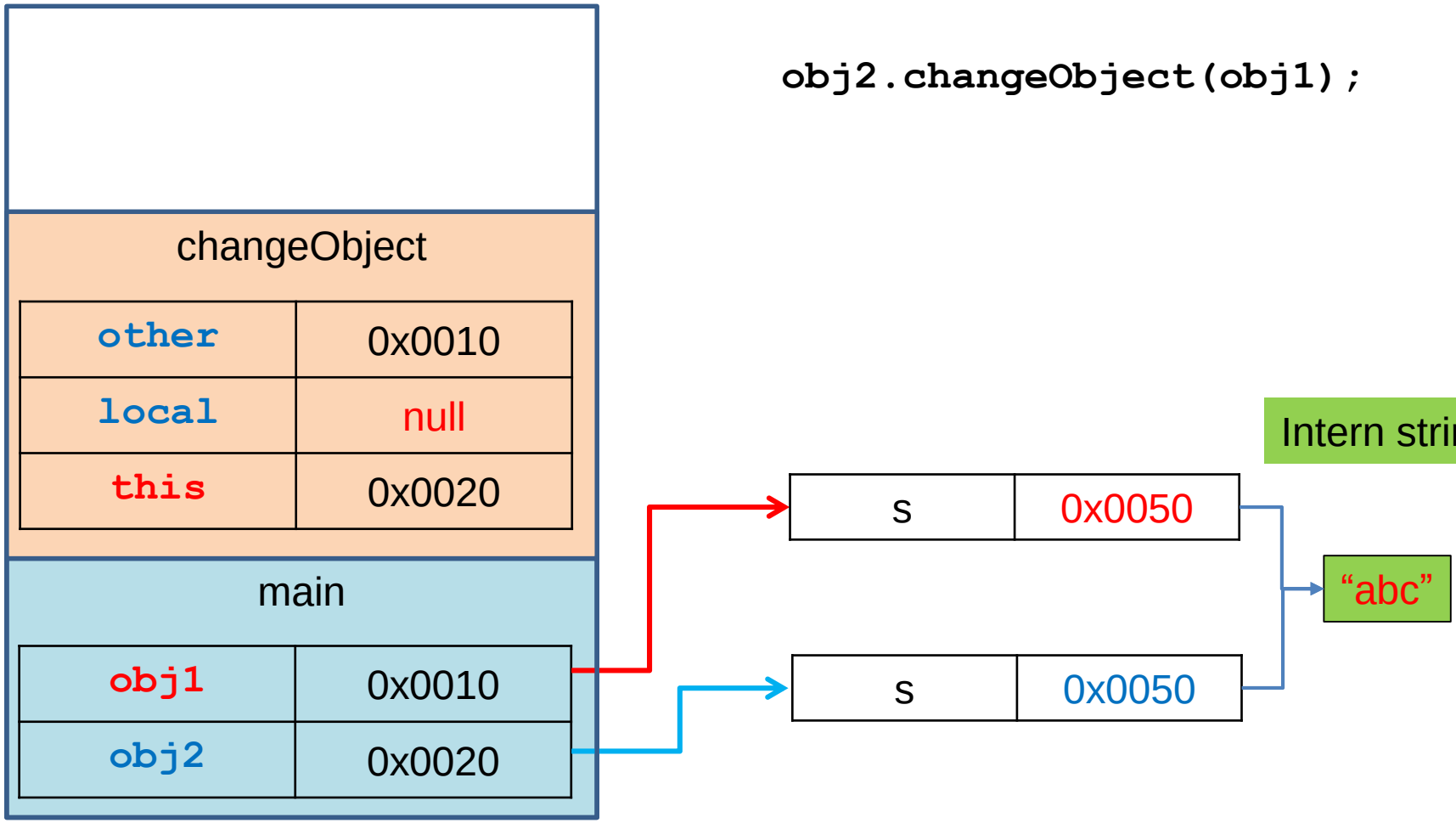


Επιστρέφοντας από την μέθοδο increment δεν υπάρχουν αλλαγές στη μεταβλητή x. .

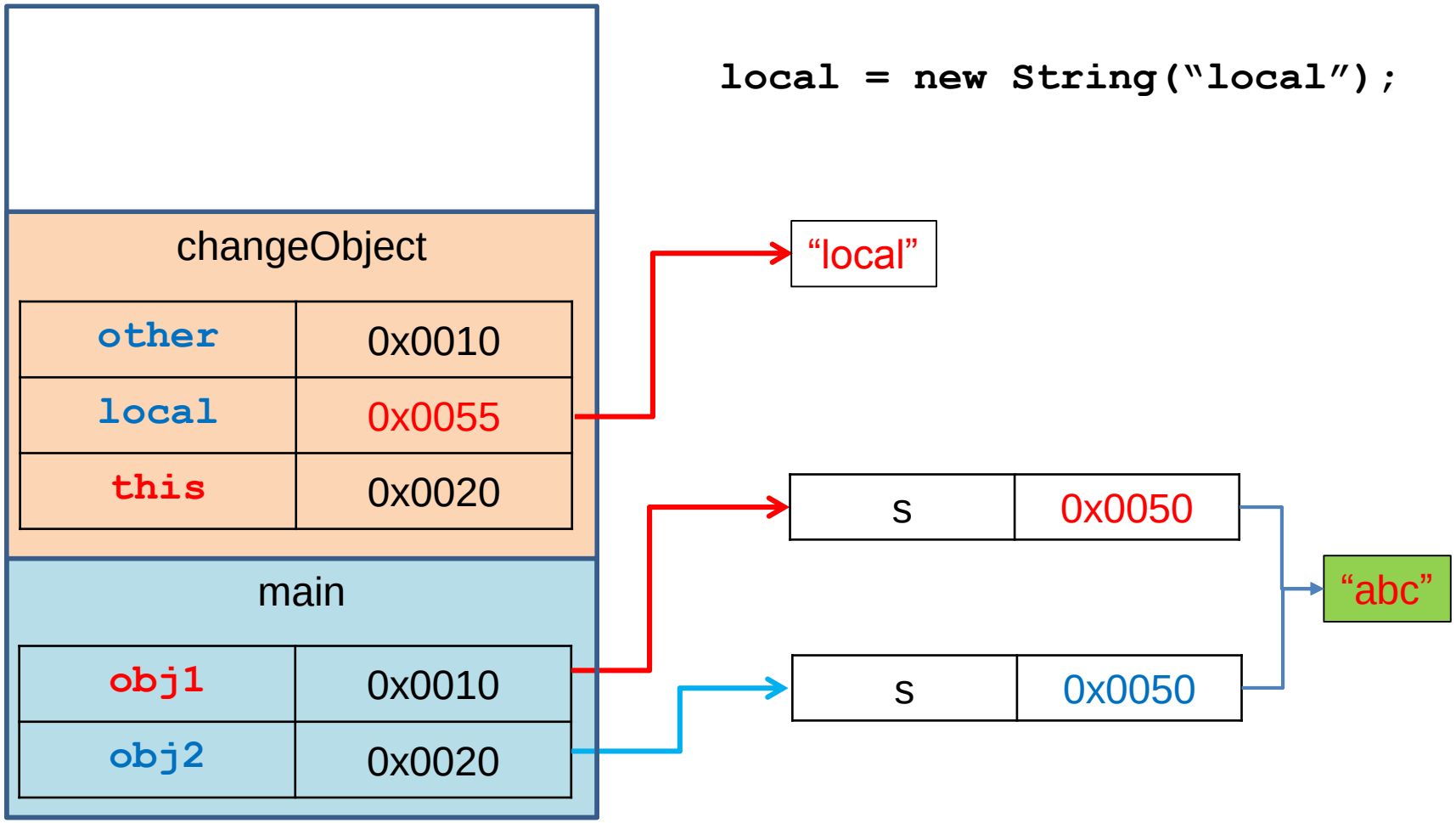
```
class ClassWithStrings
{
        String s = "abc";

        public void changeObject(ClassWithStrings other){
                String local = new String("local");
                other.s = local;
                local = "new";
                s = local;
        }

        public String toString(){
                return s;
        }
}

class StringTest
{
        public static void main(String[] args){
                ClassWithStrings obj1 = new ClassWithStrings();
                ClassWithStrings obj2 = new ClassWithStrings();
                obj2.changeObject(obj1);
                System.out.println(obj1);
                System.out.println(obj2);
        }
}
```

Τι θα τυπώσει?

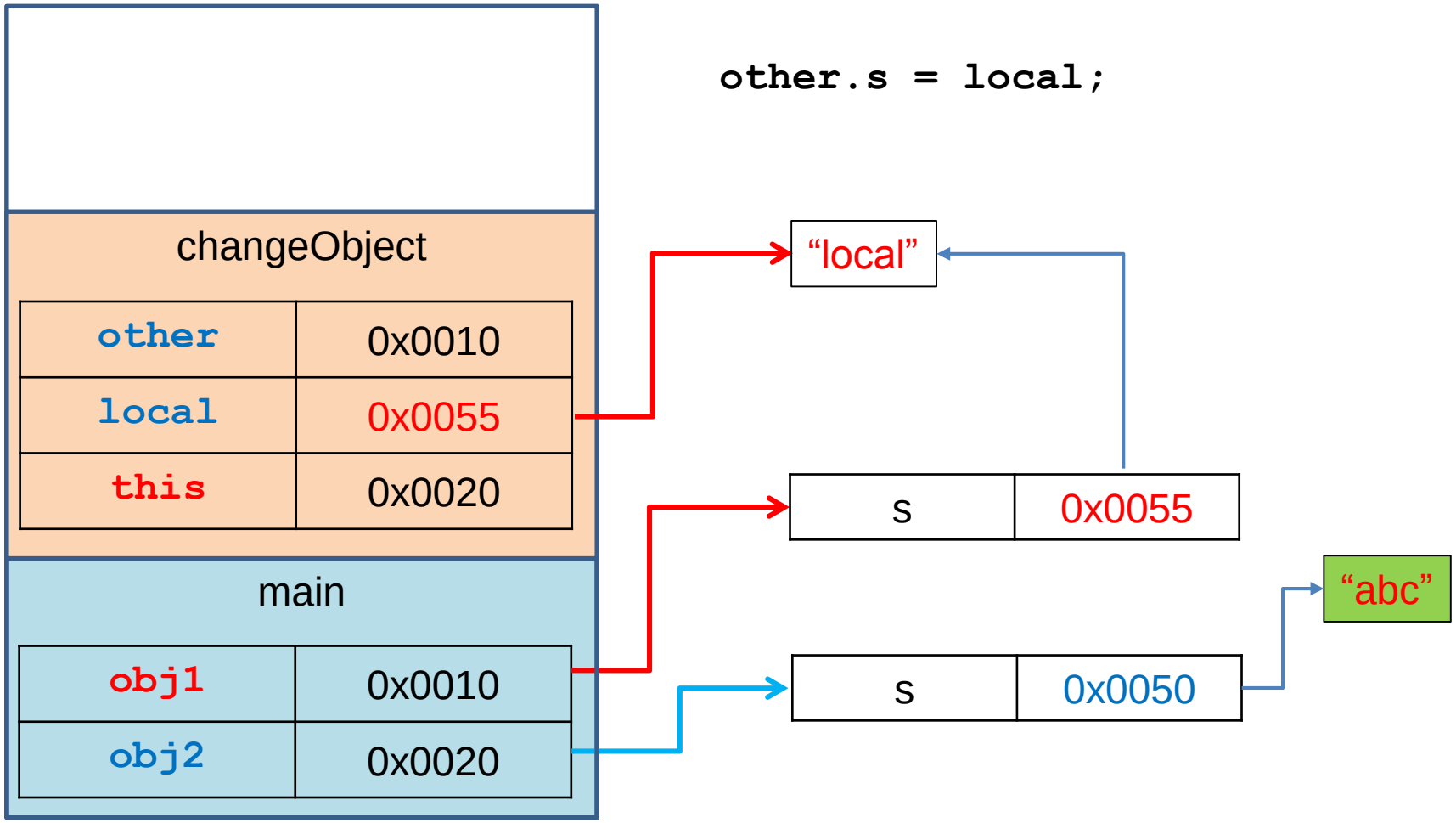
Εξέλιξη του προγράμματος



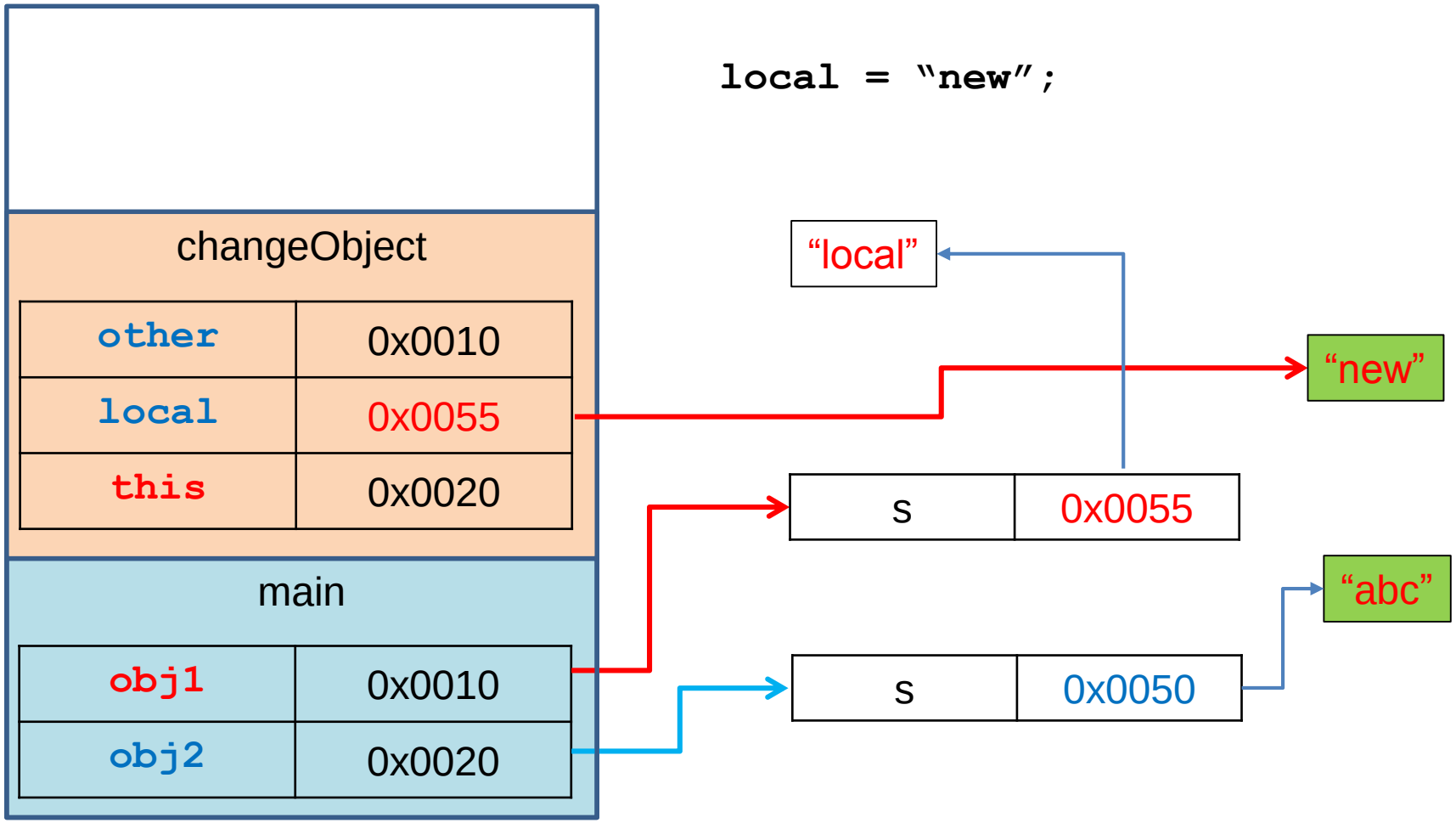
Εξέλιξη του προγράμματος



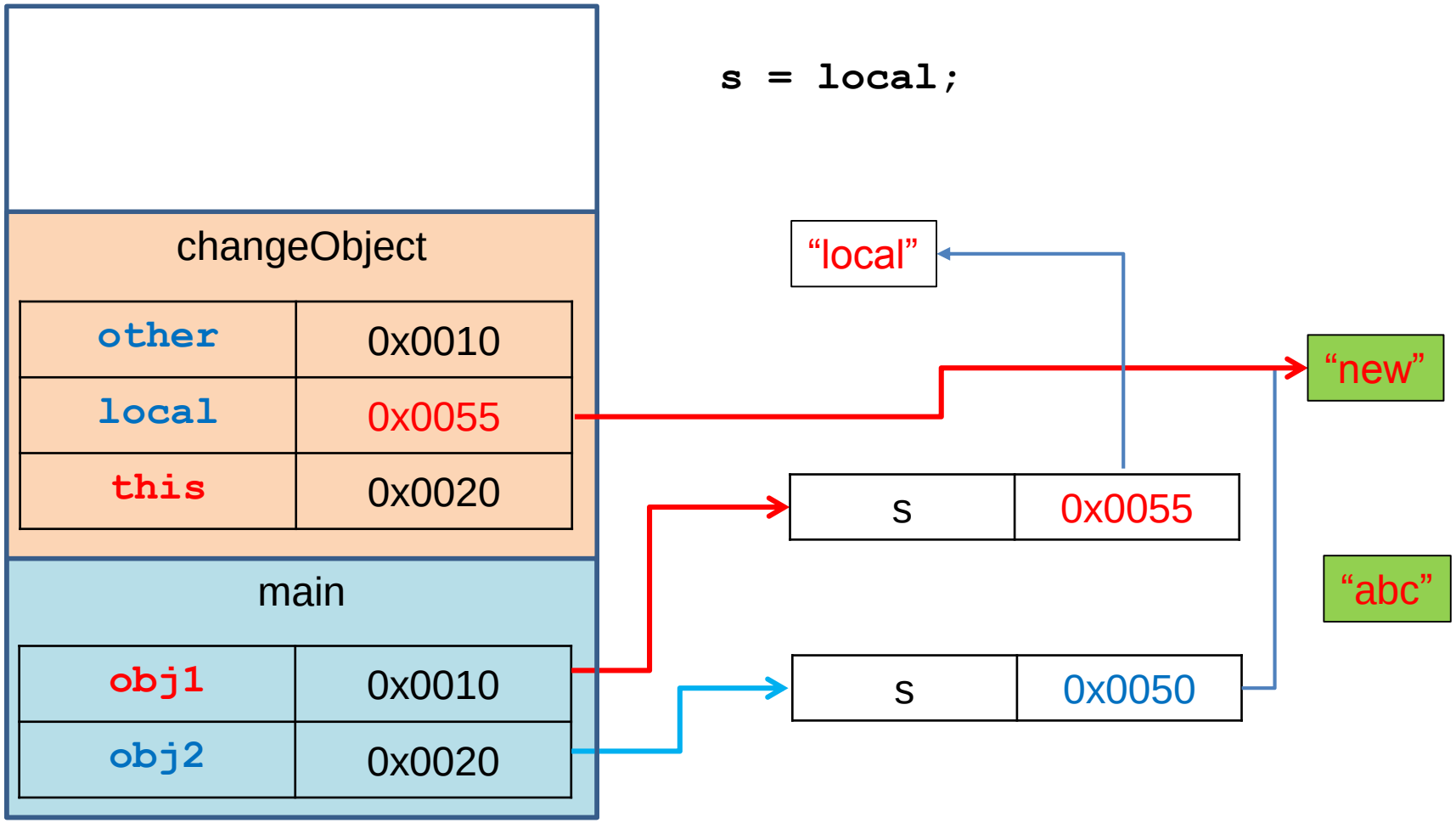
Εξέλιξη του προγράμματος



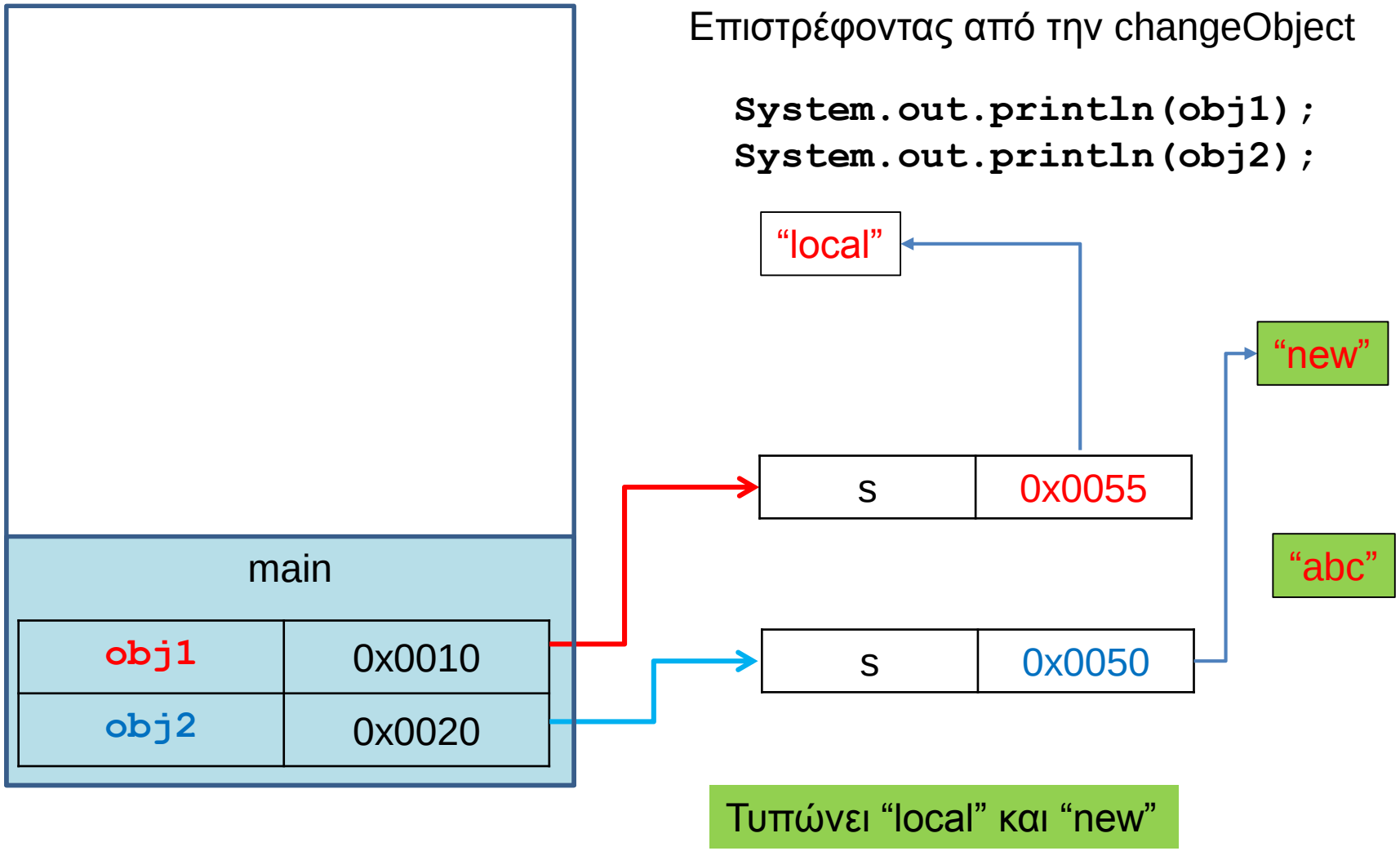
Εξέλιξη του προγράμματος



Εξέλιξη του προγράμματος



Εξέλιξη του προγράμματος



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.

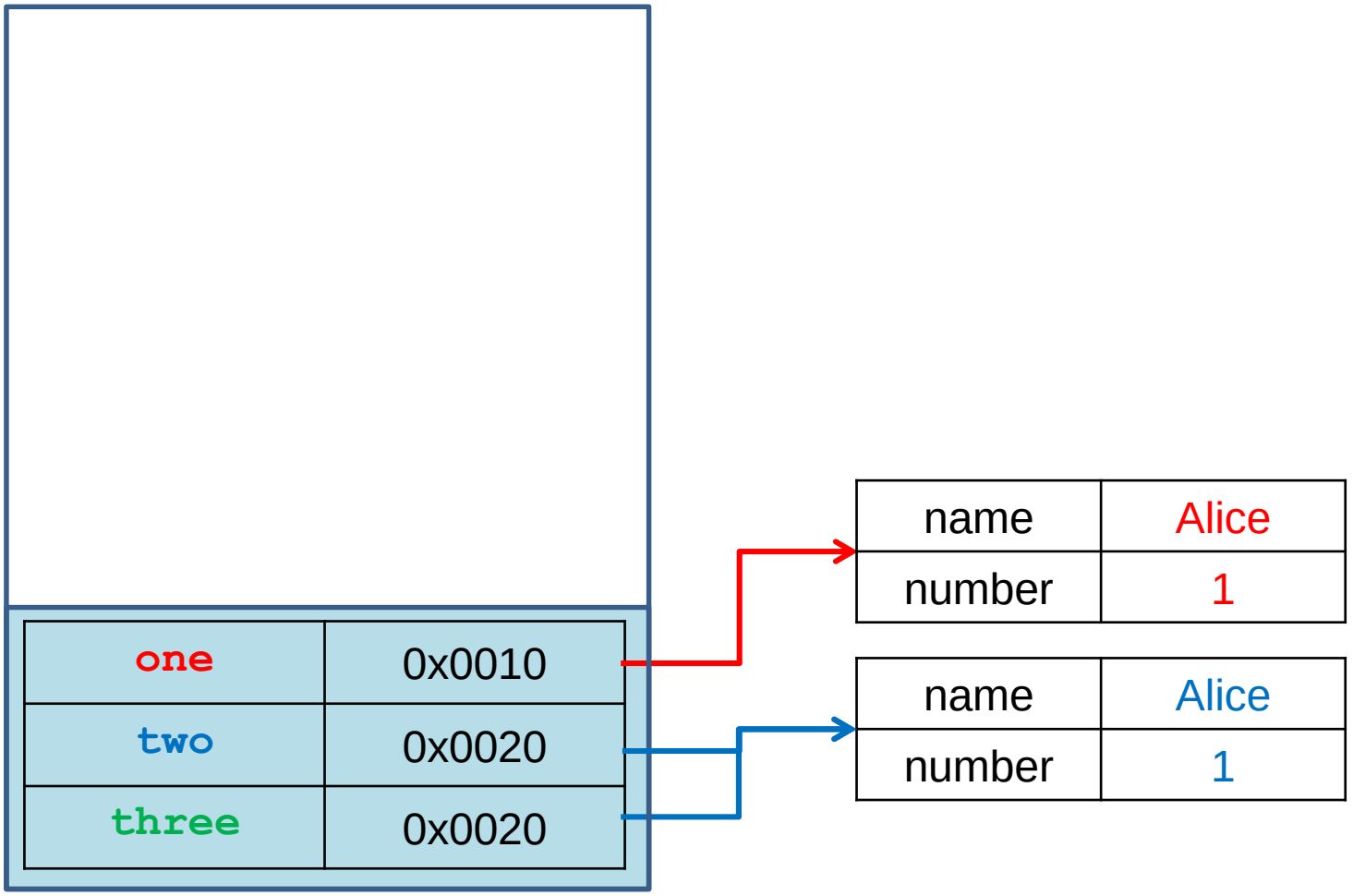
Παράδειγμα

- 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 ClassWithStrings
{
        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");
                }
        }

}

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

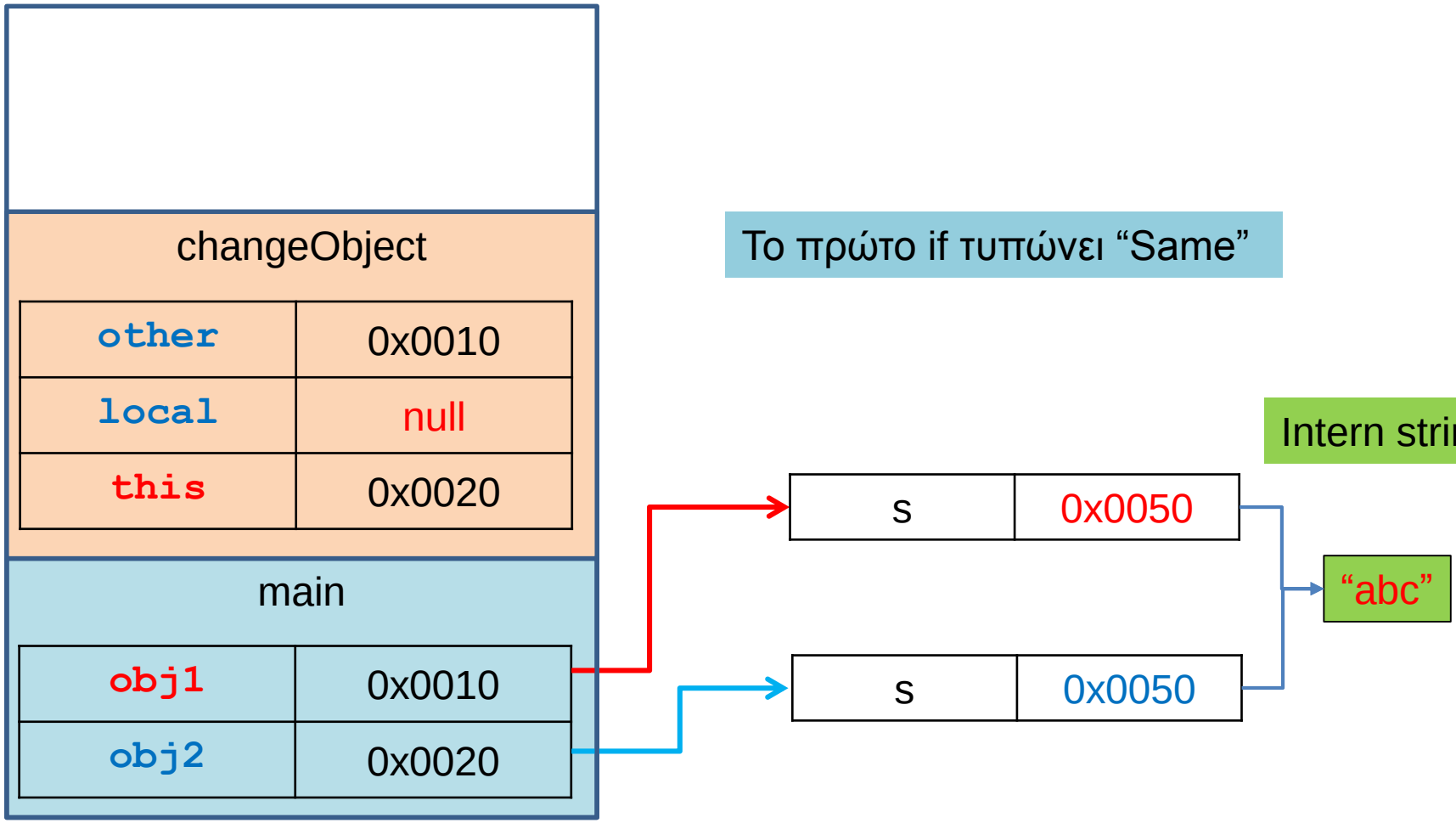
Η ανάθεση String σταθεράς έχει αποτέλεσμα την δημιουργία ενός intern string στο οποίο δείχνουν όλα τα strings στα οποία ανατίθεται η σταθερά.

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

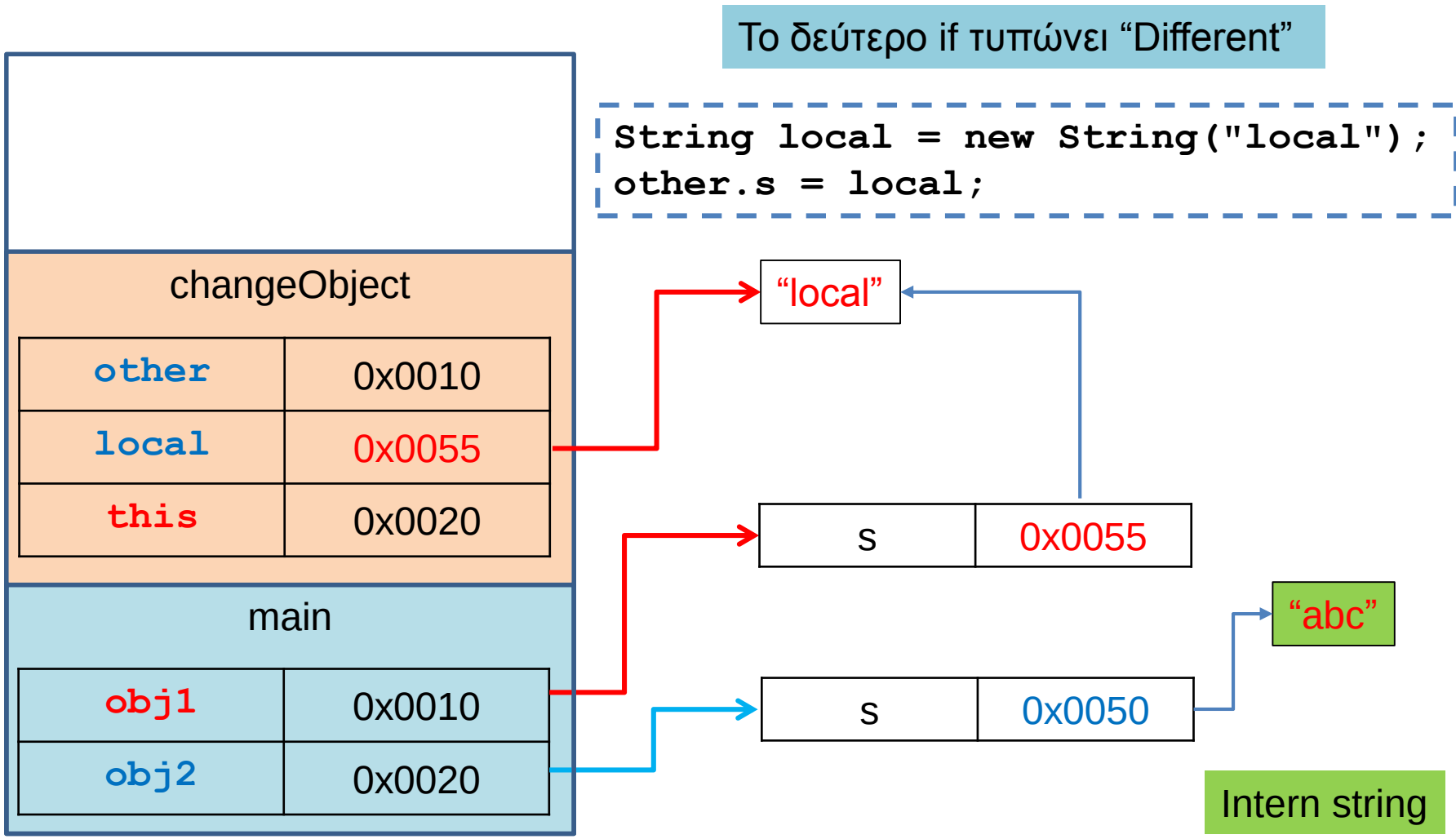
Η σταθερά δημιουργεί ένα νέο intern String

Τι θα τυπώσει?

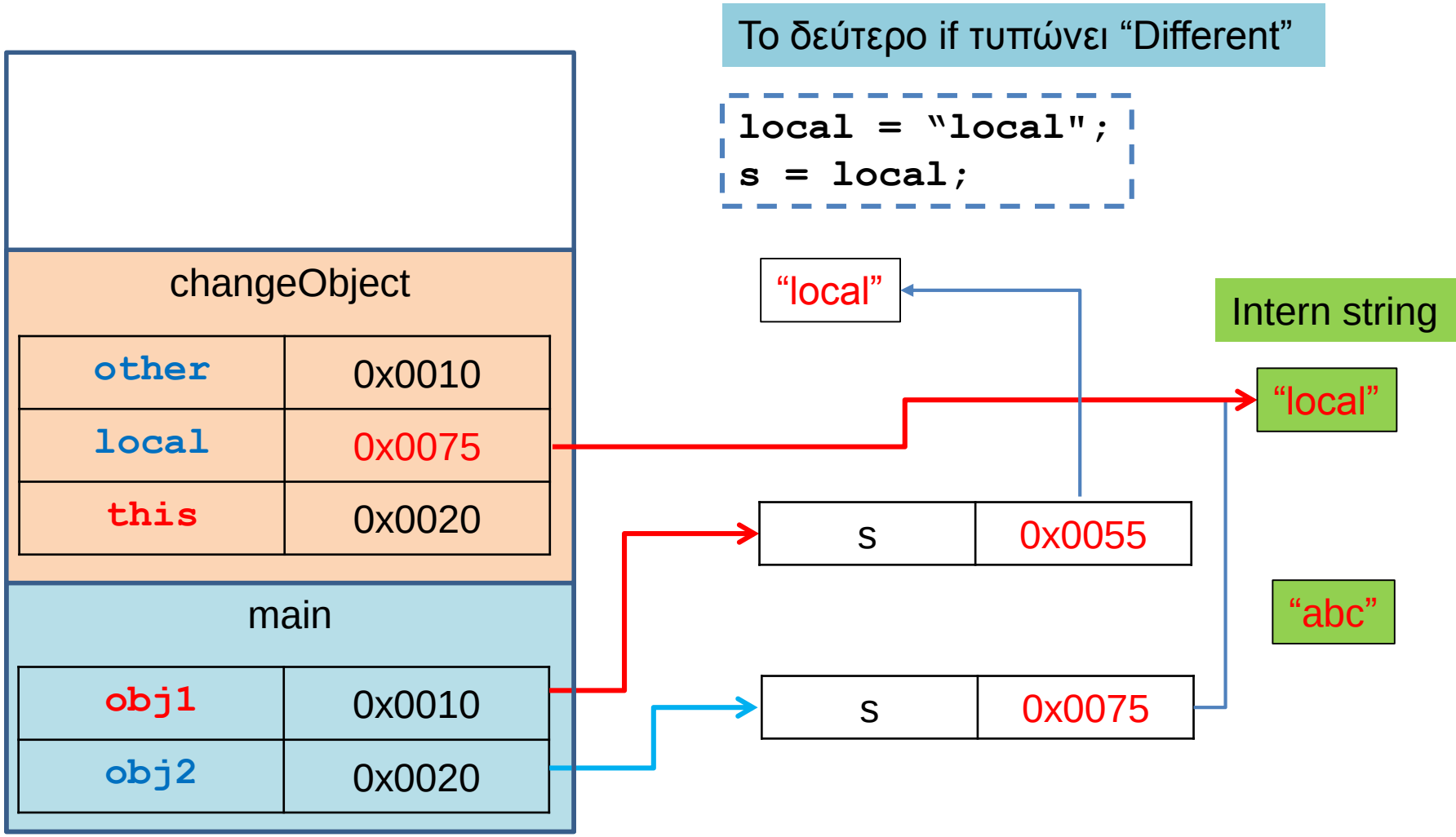
Εξέλιξη του προγράμματος



Εξέλιξη του προγράμματος



Εξέλιξη του προγράμματος



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

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

Παράδειγμα

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

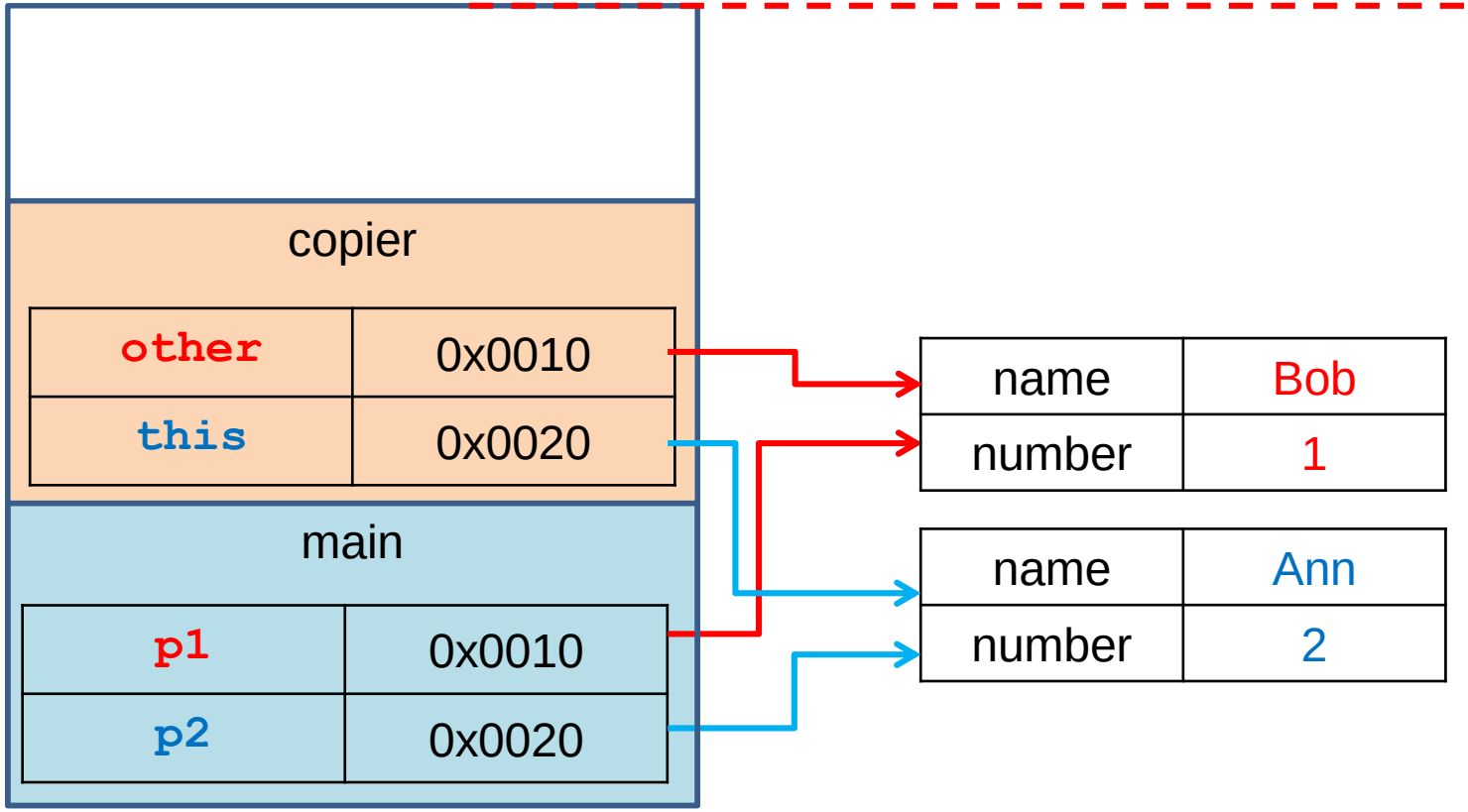
```
public class ClassParameterDemo
{
    public static void main(String[] args)
    {
        Person p1 = new Person("Bob", 1);
        Person p2 = new Person("Ann", 2);
        p2.copier(p1);
        System.out.println(p1);
     }
}
```

Τι θα τυπώσει?

Εξέλιξη του προγράμματος

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

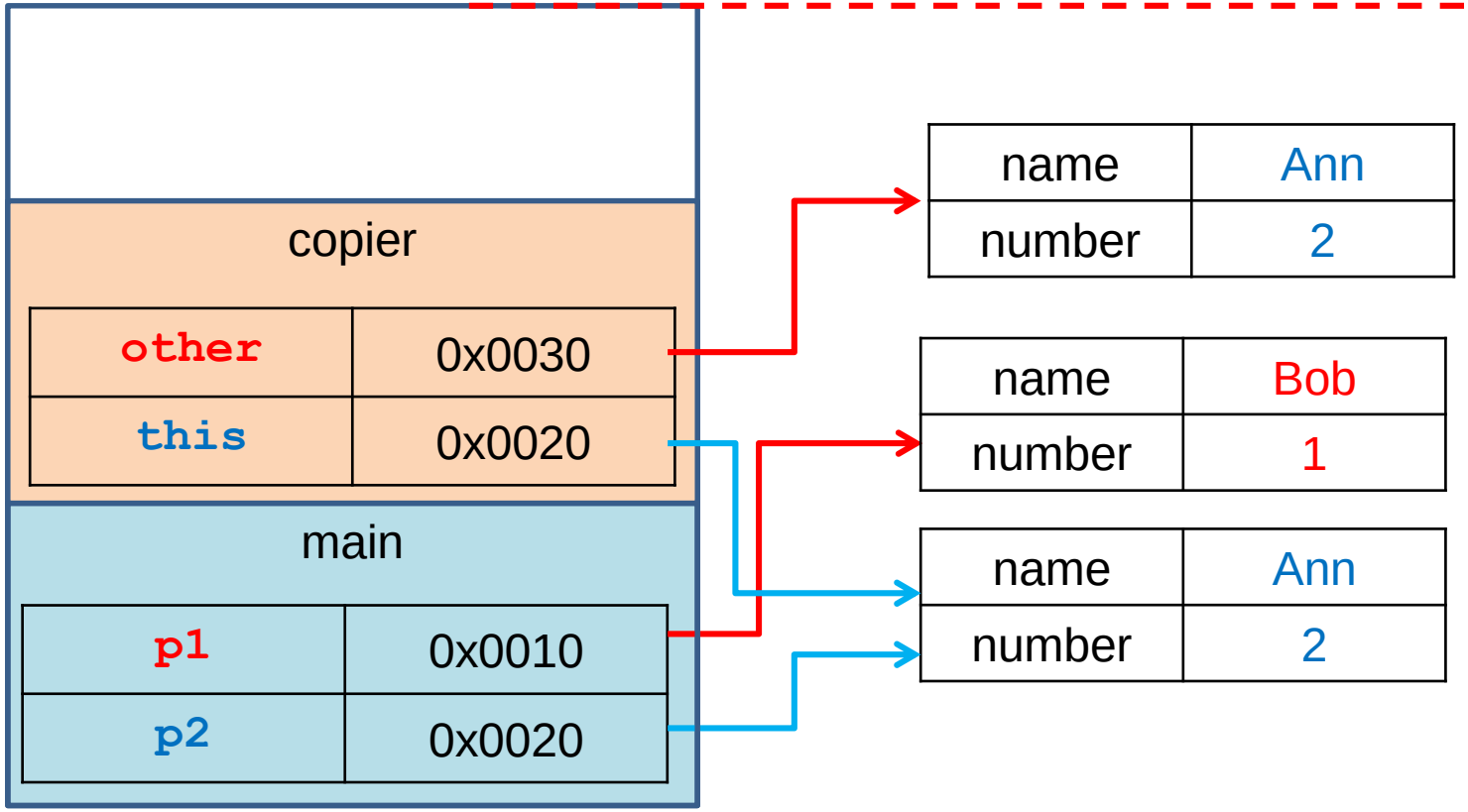
```
public void copier( Person other) {
  other = new Person(this.name, this.number);
}
```



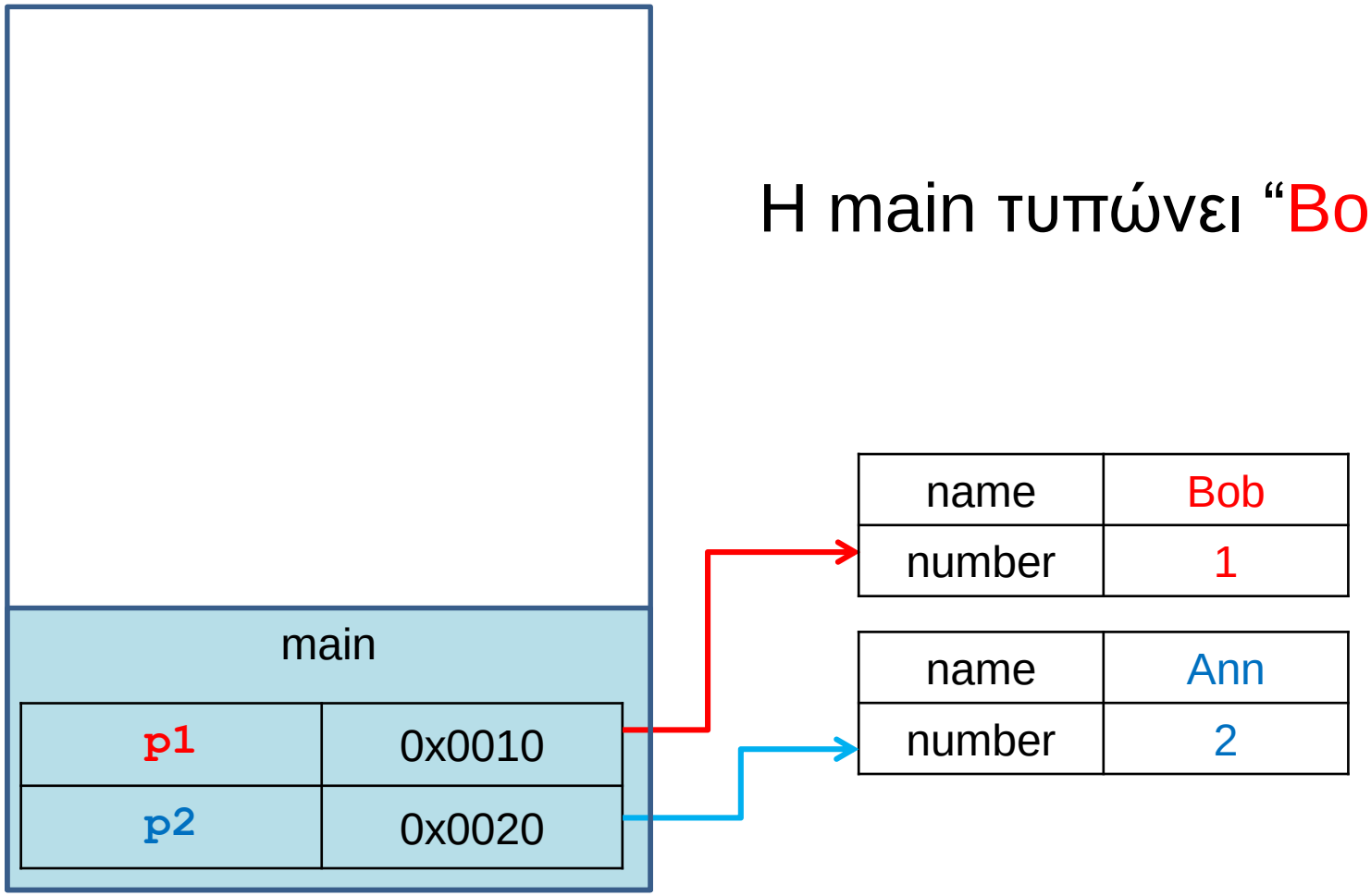
Εξέλιξη του προγράμματος

```
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.

Επιστροφή αντικειμένων

- Ένα αντικείμενο που δημιουργούμε μέσα σε μία μέθοδο μπορούμε να το διατηρήσουμε και μετά το τέλος της μεθόδου αν κρατήσουμε μια αναφορά σε αυτό.
- Ένας τρόπος να γίνει αυτό είναι αν η μέθοδος επιστρέφει το αντικείμενο (δηλαδή την αναφορά σε αυτό) που δημιουργήσαμε

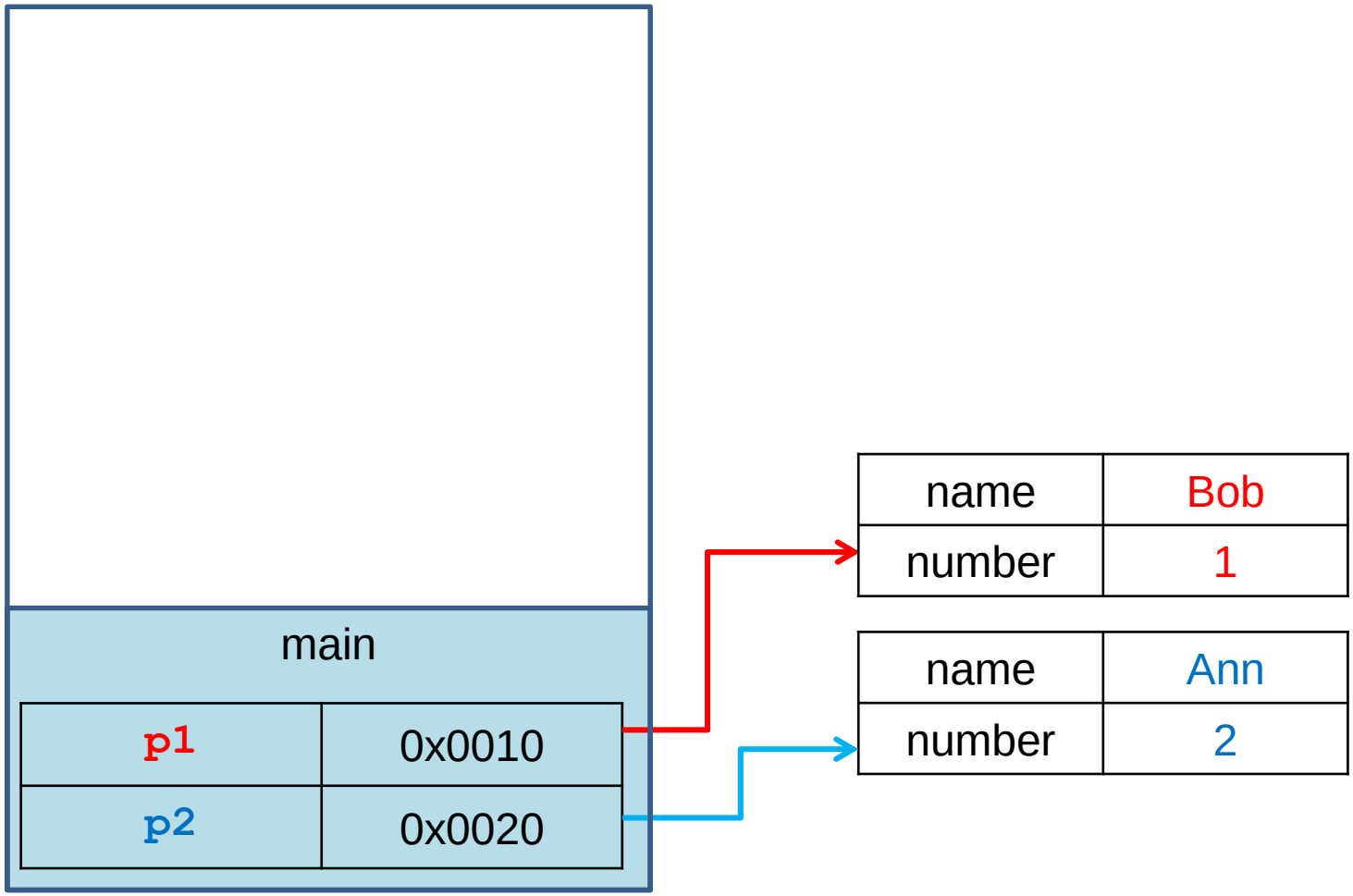
```
public class Person
{
    private String name;
    private int number;

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

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

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

    public Person copier() {
        Person newPerson = new Person(this.name,this.number);
        return newPerson;
    }

}
```

Παράδειγμα

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

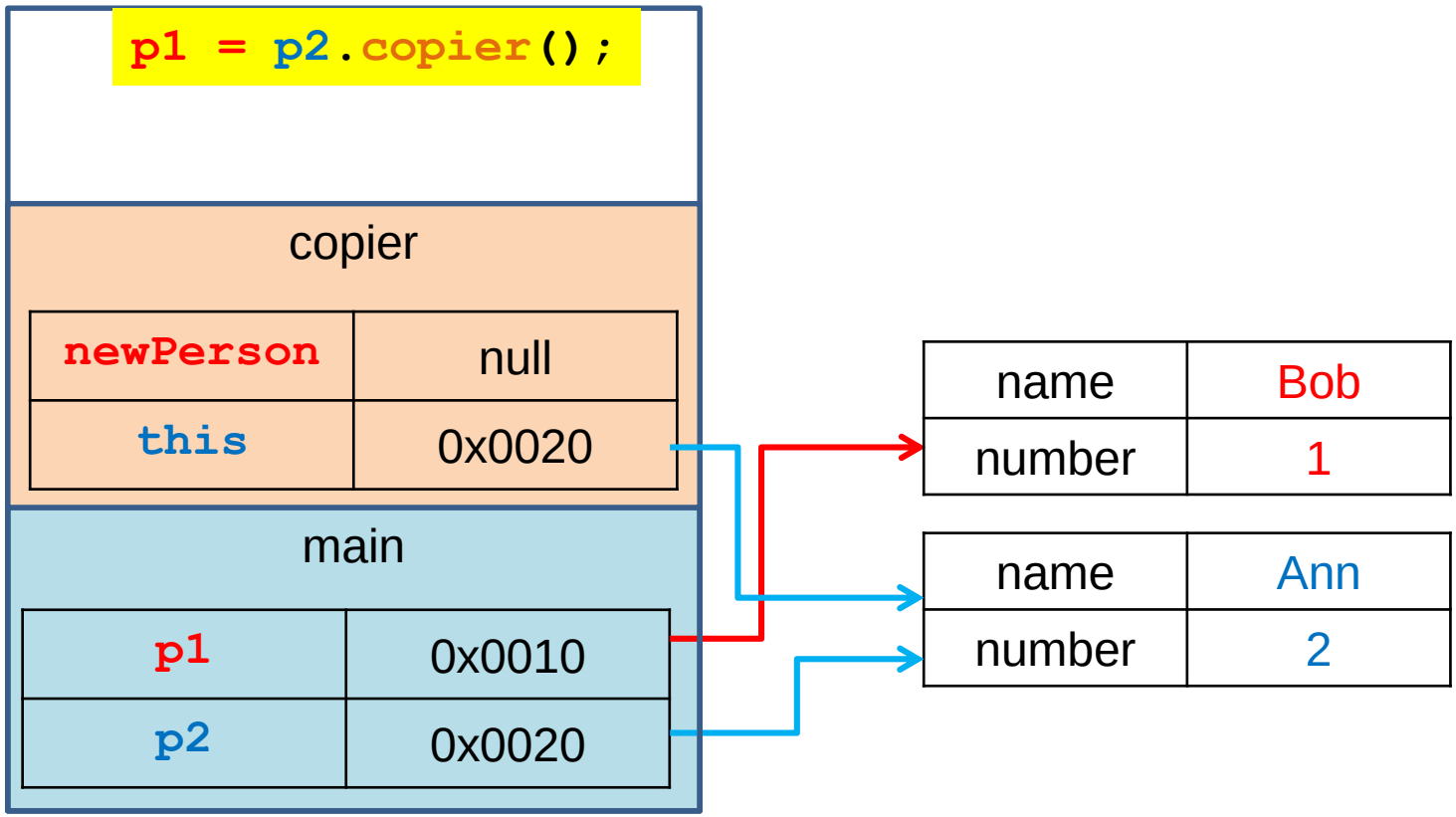
Τι θα τυπώσει?

Εξέλιξη του προγράμματος



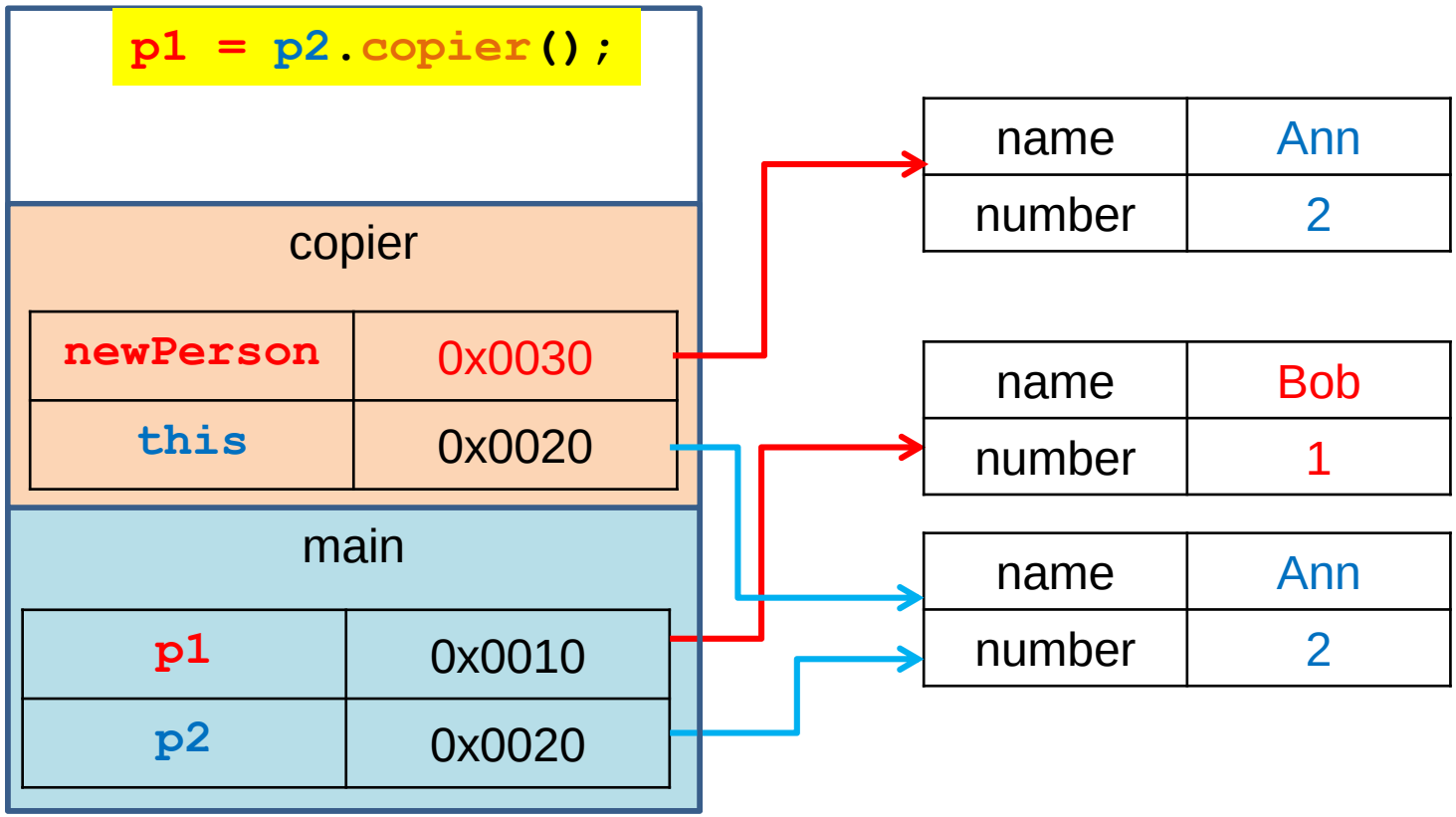
Εξέλιξη του προγράμματος

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



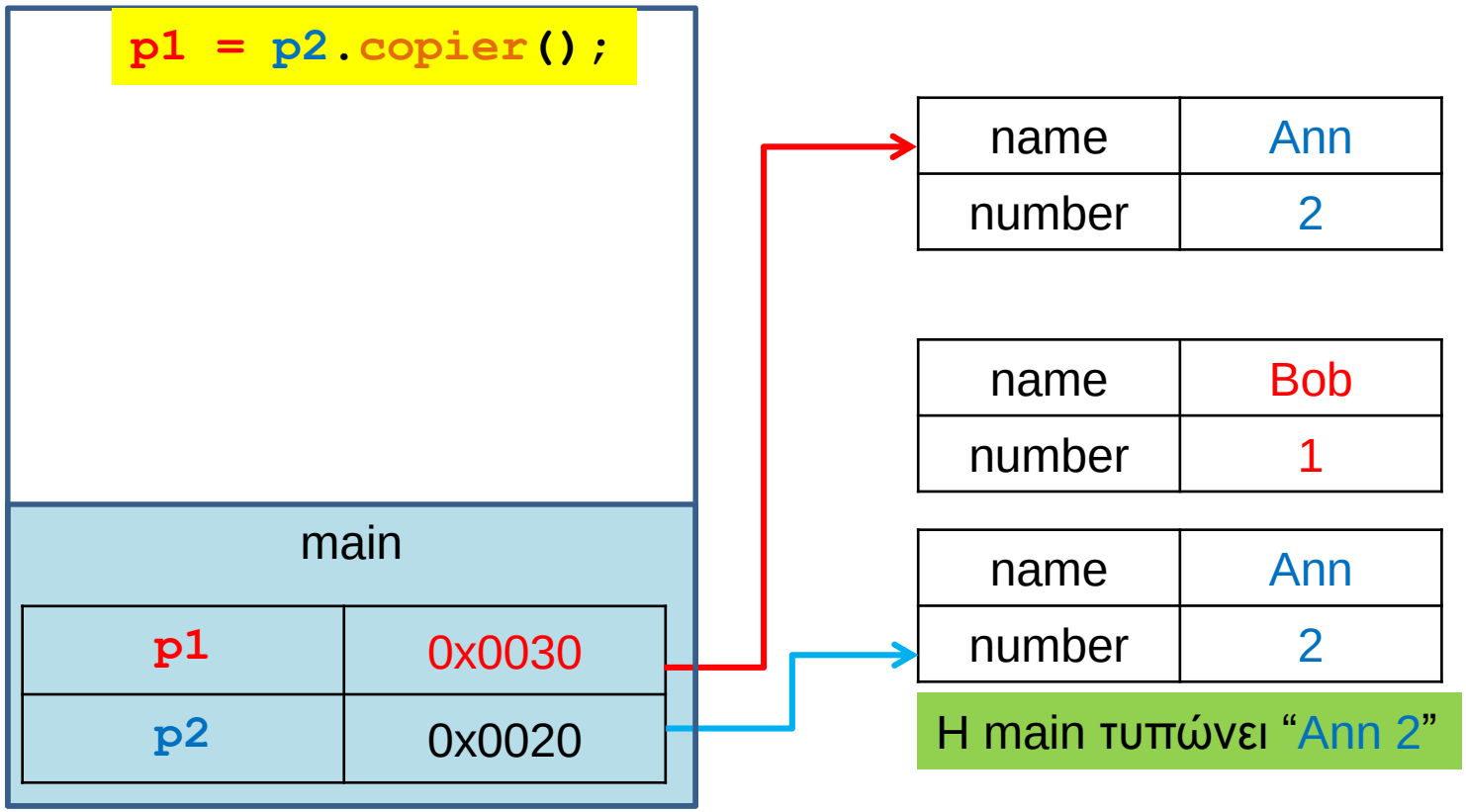
Εξέλιξη του προγράμματος

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



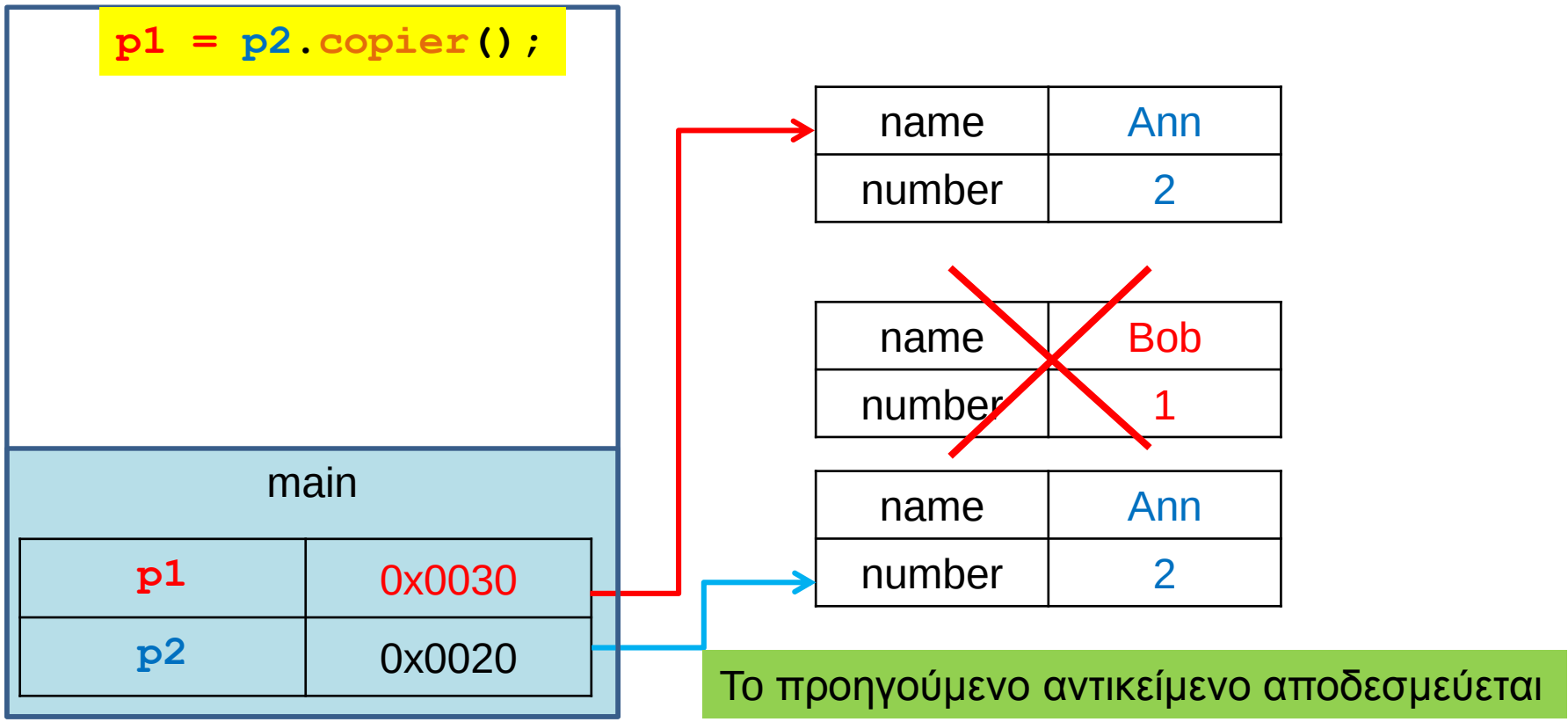
Εξέλιξη του προγράμματος

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



Εξέλιξη του προγράμματος

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



Δημιουργία αντιγράφων

- Η μέθοδος copier όπως την ορίσαμε πριν δημιουργεί ένα καινούριο αντικείμενο που είναι αντίγραφο αυτού που έκανε την κλήση.
- Στην περίπτωση μας το αντικείμενο έχει μόνο πεδία που είναι πρωταρχικού τύπου ή μη μεταλλάξιμα αντικείμενα. Γενικά ένα αντικείμενο μπορεί να έχει ως πεδία άλλα αντικείμενα (δηλαδή αναφορές).
- Στην περίπτωση αυτή η δημιουργία αντιγράφου θα πρέπει να γίνεται με πολύ προσοχή!

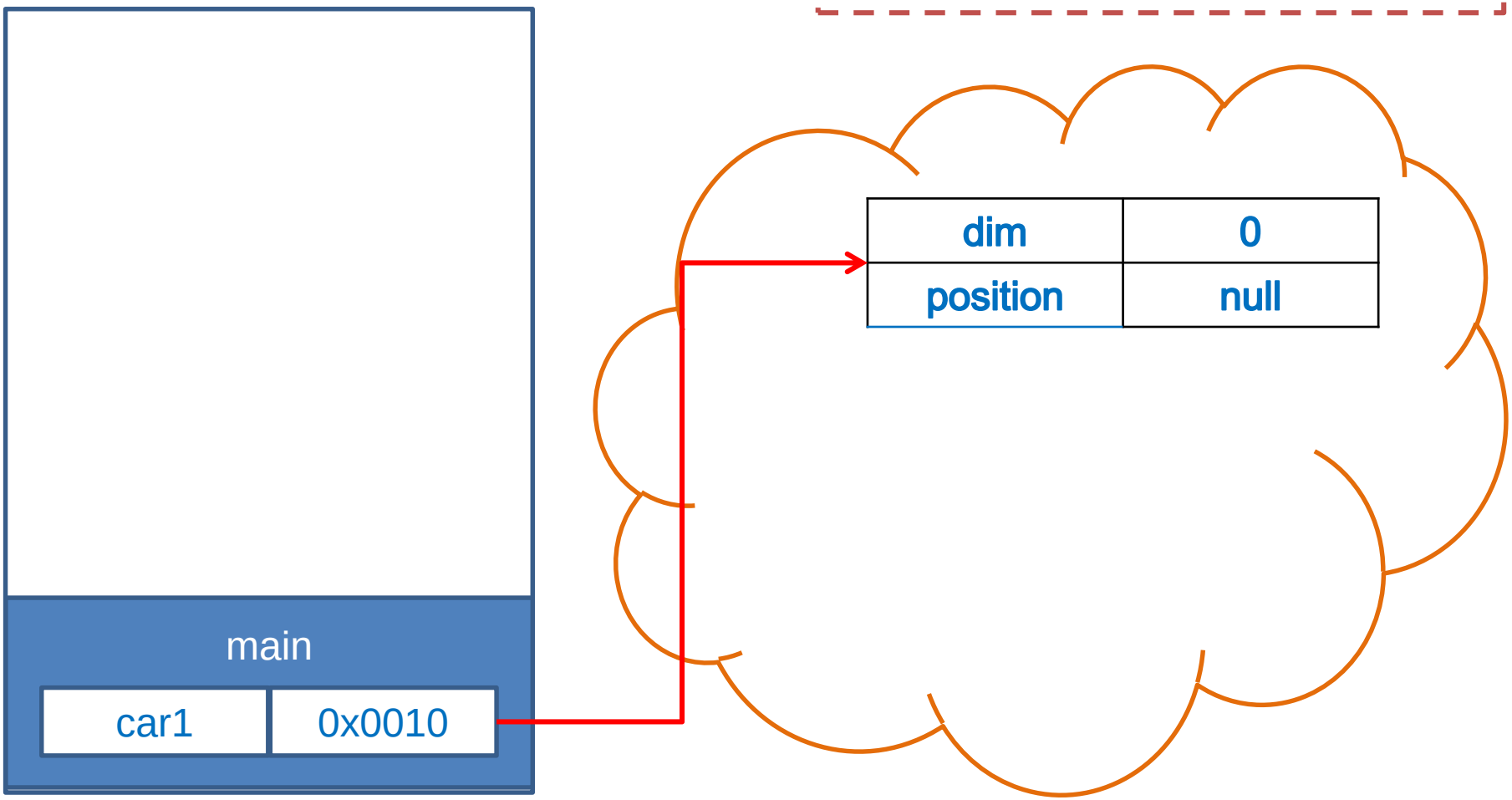
```
class Car
{
    private int[] position;
    private int dim;

    public Car(int d){
       dim = d;
       position = new int[d];
    }

    public void move(){
       for (int i=0; i < dim; i++){
          position[i] ++;
       }
   }

    public String toString(){
       String output = "";
       for (int i=0; i < dim; i++){
           output = output + position[i] + " ";
       }
       return output;
    }

    public static void main(String args[]){
          Car car1 = new Car(2);
          car1.move();
          System.out.println(car1);
    }
}
```

Ένα όχημα που κινείται σε πολλές διαστάσεις

Τι γίνεται όταν δημιουργούμε ένα αντικείμενο Car?

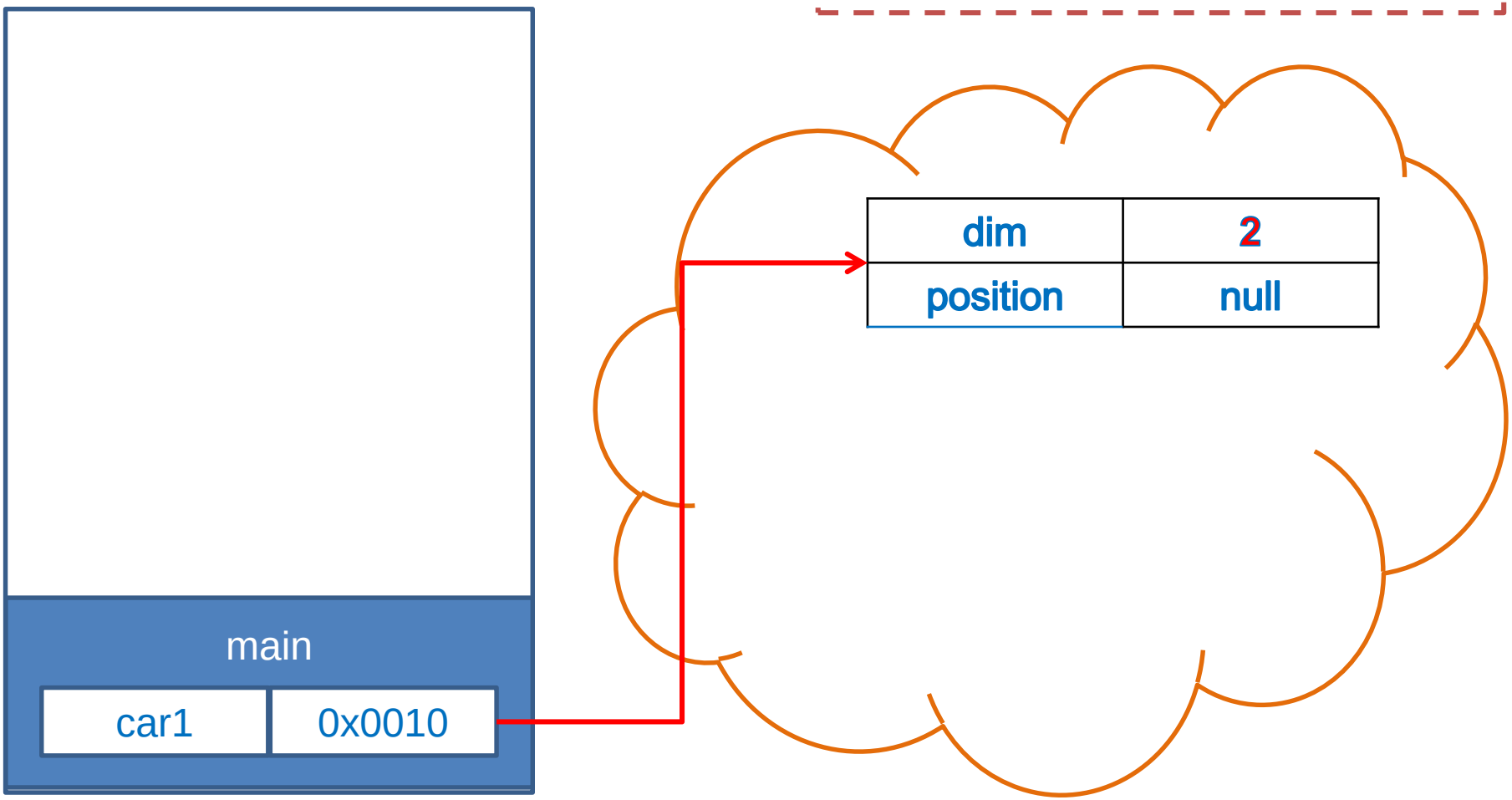
```
public void main()
{
   Car car1 = new Car(2)
}
```

```
public Car(int d)
{
    dim = d;
    position = new int[dim];
}
```



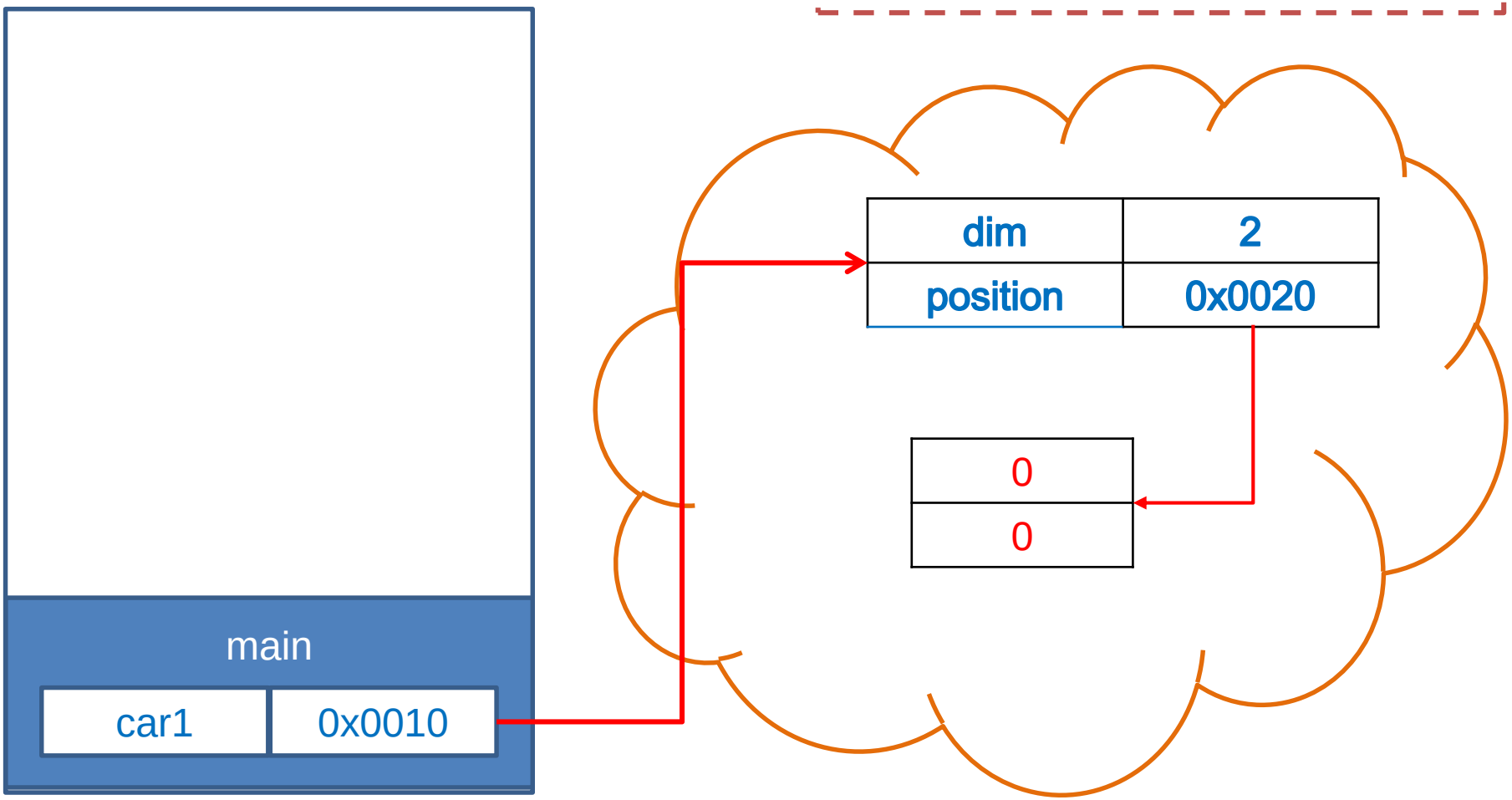
```
public void main()
{
   Car car1 = new Car(2)
}
```

```
public Car(int d)
{
    dim = d;
    position = new int[dim];
}
```



```
public void main()
{
   Car car1 = new Car(2)
}
```

```
public Car(int d)
{
    dim = d;
    position = new int[dim];
}
```



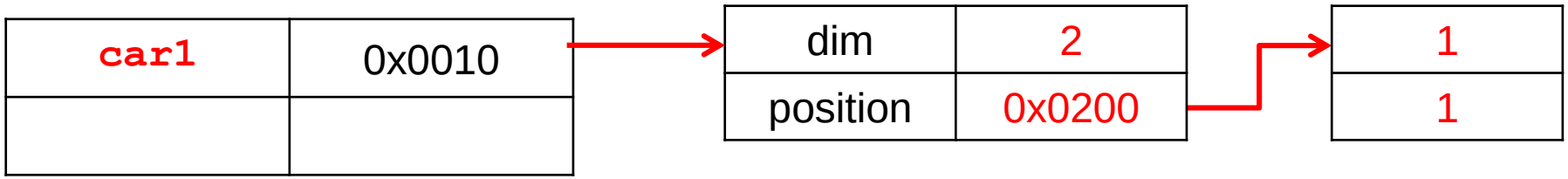
```
class Car
{
    private int[] position;
    private int dim;

    public Car(int d){
       dim = d;
       position = new int[d];
    }

    public void move(){
       for (int i=0; i < dim; i++){
          position[i] ++;
       }
   }

    public Car copy(){
       Car newCar = new Car(this.dim);
       newCar.position = this.position;
       return newCar;
    }

    public String toString(){
       String output = "";
       for (int i=0; i < dim; i++){
           output = output + position[i] + " ";
       }
       return output;
    }

    public static void main(String args[]){
           Car car1 = new Car(2);
           car1.move();
           Car car2 = car1.copy();
           car2.move();
           System.out.println(car1);
    }
}
```

Η copy δημιουργεί και επιστρέφει ένα νέο Car

Τι θα τυπώσει η main?

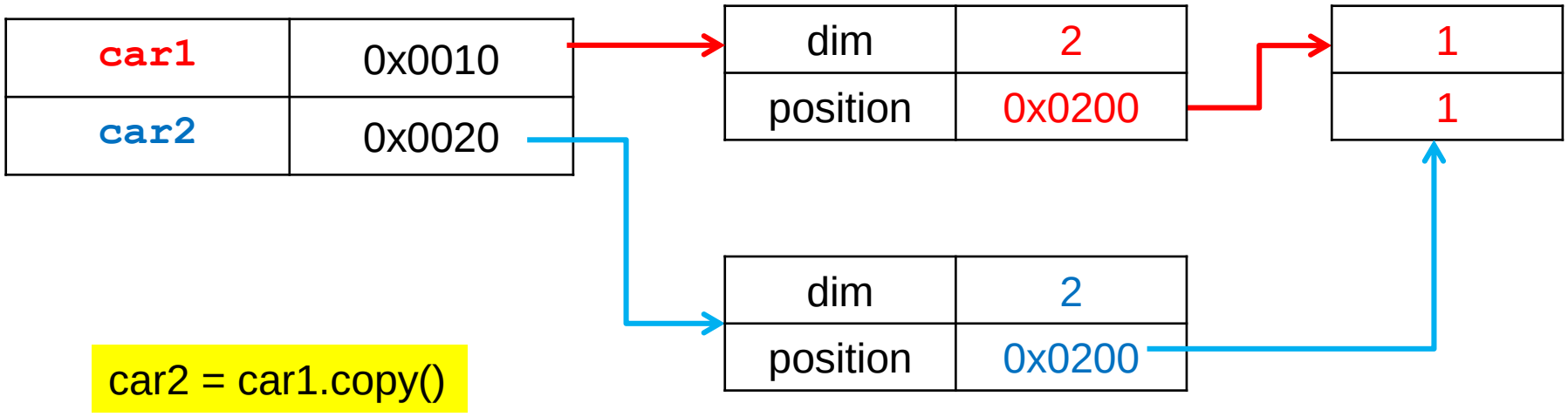
Ρηχά Αντίγραφα

- Η copy όπως την έχουμε ορίσει δημιουργεί ένα ρηχό αντίγραφο του αντικειμένου
  - Αντιγράφει τις αναφορές στα αντικείμενα και όχι τα περιεχόμενα των αντικειμένων



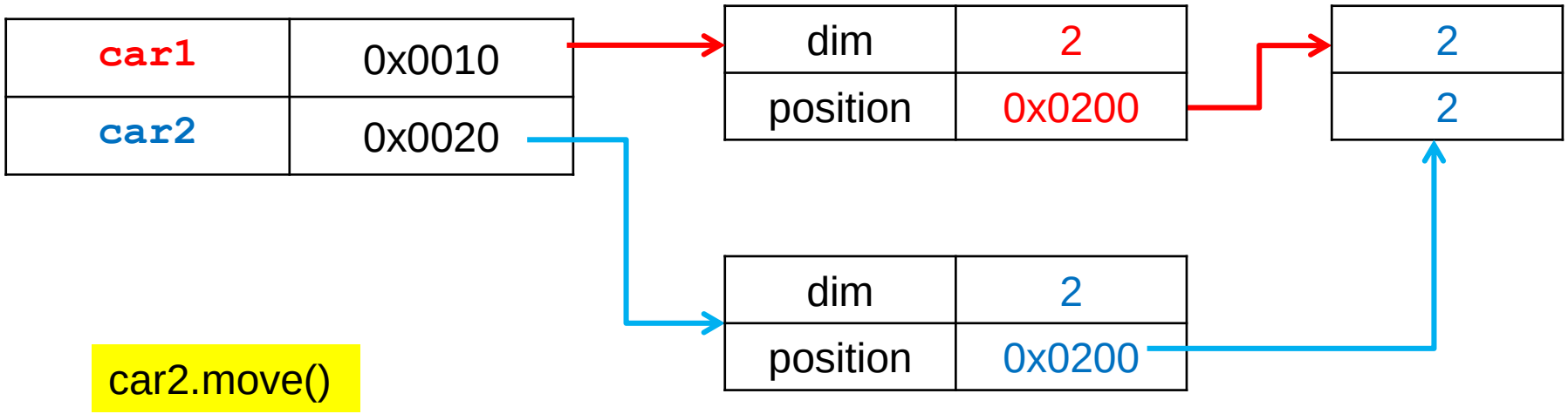
Ρηχά Αντίγραφα

- Η copy όπως την έχουμε ορίσει δημιουργεί ένα ρηχό αντίγραφο του αντικειμένου
  - Αντιγράφει τις αναφορές στα αντικείμενα και όχι τα περιεχόμενα των αντικειμένων



Ρηχά Αντίγραφα

- Η copy όπως την έχουμε ορίσει δημιουργεί ένα ρηχό αντίγραφο του αντικειμένου
  - Αντιγράφει τις αναφορές στα αντικείμενα και όχι τα περιεχόμενα των αντικειμένων

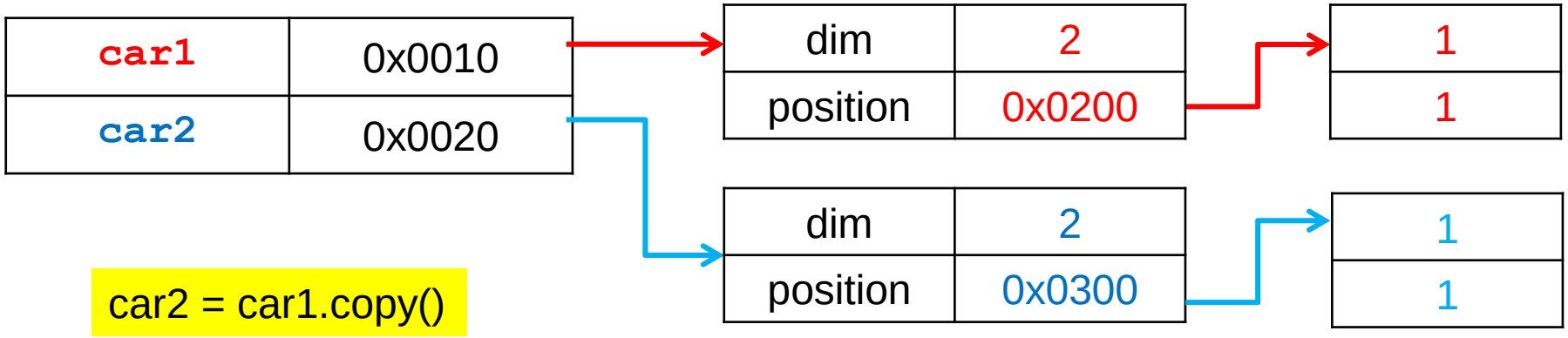


Μετακινείται και το car1 αλλά αυτό δεν είναι επιθυμητό.

Βαθύ αντίγραφο

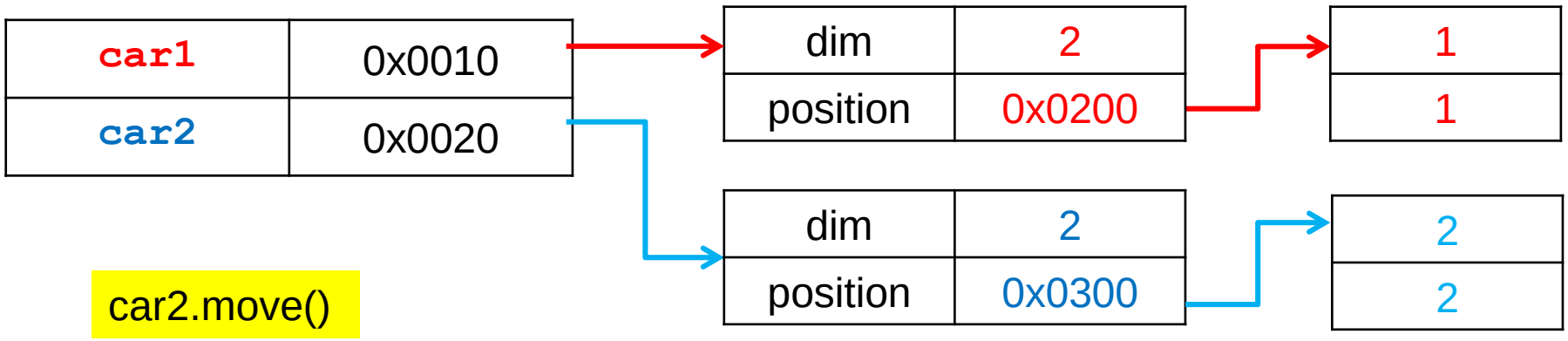
- Τις περισσότερες φορές θέλουμε να κάνουμε ένα βαθύ αντίγραφο του αντικειμένου, όπου για κάθε αντικείμενο μέσα στο αντίγραφο δεσμεύουμε νέα μνήμη

```
public Car copy(){
    Car newCar = new Car(this.dim);
    for (int i=0; i<dim; i++){
      newCar.position[i] = this.position[i];
    }
    return newCar;
}
```



Βαθύ αντίγραφο

- Το βαθύ αντίγραφο του car1 είναι πλέον ένα ανεξάρτητο αντικείμενο.



Η μετακίνηση του car2 δεν επηρεάζει το car1