ΤΕΧΝΙΚΕΣ ΑΝΤΙΚΕΙΜΕΝΟΣΤΡΑΦΟΥΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ

Αναφορές

Στοίβα και Σωρός Μνήμης

Αντικείμενα ως ορίσματα

ΑΝΑΦΟΡΕΣ

new

- Όπως είδαμε για να δημιουργήσουμε ένα αντικείμενο χρειάζεται να καλέσουμε τη new.
  - Για τον πίνακα είπαμε ότι έτσι δίνουμε χώρο στον πίνακα και δεσμεύουμε την απαιτούμενη μνήμη.

- Τι ακριβώς συμβαίνει όταν καλούμε την new?

Η μνήμη του υπολογιστή

- Η κύρια μνήμη (main memory) του υπολογιστή κρατάει τα δεδομένα (και τις εντολές) για την εκτέλεση των προγραμμάτων.
  - Η μνήμη είναι προσωρινή, τα δεδομένα χάνονται όταν ολοκληρωθεί το πρόγραμμα.
- Η μνήμη είναι χωρισμένη σε bytes (8 bits)
  - Ο χώρος που χρειάζεται για ένα χαρακτήρα ASCII.
- Το κάθε byte έχει μια διεύθυνση, με την οποία μπορούμε να προσπελάσουμε τη συγκεκριμένη θέση μνήμης
  - Random Access Memory (RAM)
  - Σε 32-bit συστήματα μια διεύθυνση είναι 32 bits, σε 64-bit συστήματα μια διεύθυνση είναι 64 bits.

Διεύθυνση μνήμης	Περιεχόμενο μνήμης
0000	‘a’
0001	‘b’
0010	‘c’
0011	‘d’
0100	‘e’
0101	‘f’
0110	‘g’
0111	‘h’

Αποθήκευση μεταβλητών

- Η κύρια μνήμη (main memory) του υπολογιστή κρατάει τις μεταβλητές ενός προγράμματος
- Μια μεταβλητή μπορεί να απαιτεί χώρο περισσότερο από 1 byte.
  - Π.χ., οι μεταβλητές τύπου double χρειάζονται 8 bytes.
  - Η μεταβλητή τότε αποθηκεύεται σε συνεχόμενα bytes στη μνήμη.
- Η θέση μνήμης (διεύθυνση) της μεταβλητής θεωρείται το πρώτο byte από το οποίο ξεκινάει η αποθήκευση του της μεταβλητής.
  - Στο παράδειγμα μας η μεταβλητή βρίσκεται στη θέση 0000
  - Αν ξέρουμε την αρχή και το μέγεθος της μεταβλητής μπορούμε να τη διαβάσουμε.
- Άρα μία θέση μνήμης αποτελείται από μία διεύθυνση και το μέγεθος.

Διεύθυνση μνήμης	Περιεχόμενο μνήμης
0000	8.5
0001	
0010	
0011	
0100	
0101	
0110	
0111	

Αποθήκευση μεταβλητών πρωταρχικού τύπου

- Για τις μεταβλητές πρωταρχικού τύπου (char, int, double,…) ξέρουμε εκ των προτέρων το μέγεθος της μνήμης που χρειαζόμαστε.
- Όταν ο μεταγλωττιστής δει τη δήλωση μιας μεταβλητής πρωταρχικού τύπου δεσμεύει μια θέση μνήμης αντίστοιχου μεγέθους
  - Η δήλωση μιας μεταβλητής ουσιαστικά δίνει ένα όνομα σε μία θέση μνήμης
  - Συχνά λέμε η θέση μνήμης x για τη μεταβλητή x.

```
int x = 5;
int y = 3;
```

	Διεύθυνση μνήμης	Περιεχόμενο μνήμης
x	0000	5
	0001	
	0010	
	0011	
y	0100	3
	0101	
	0110	
	0111	

Αποθήκευση αντικειμένων

- Για τα αντικείμενα δεν ξέρουμε πάντα εκ των προτέρων το μέγεθος της μνήμης που θα πρέπει να δεσμεύσουμε.

```
String s; // δεν ξερουμε το μέγεθος του s
s = "ab"; // το s έχει μέγεθος 2 χαρακτήρες
s = "abc"; //  το s έχει μέγεθος 3 χαρακτήρες
```

- Παρομοίως αν δηλώσουμε

```
int[] A;
```

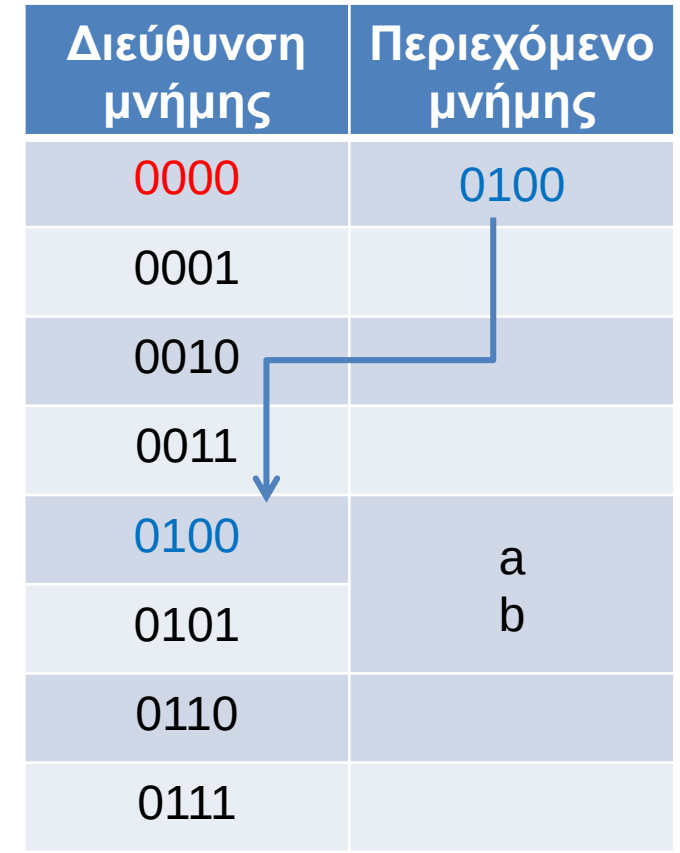
μας λέει ότι έχουμε ένα πίνακα από ακέραιους αλλά δεν μας λέει πόσο μεγάλος θα είναι αυτός ο πίνακας.

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

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];
```

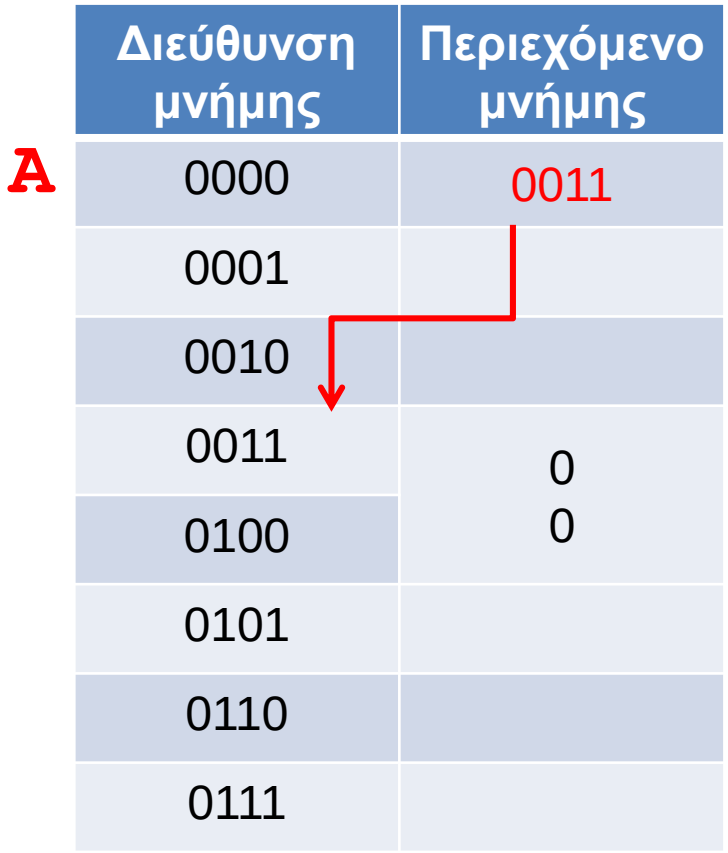
	Διεύθυνση μνήμης	Περιεχόμενο μνήμης
A	0000	null
	0001	
	0010	
	0011	
	0100	
	0101	
	0110	
	0111	

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

Παράδειγμα - πινάκες

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

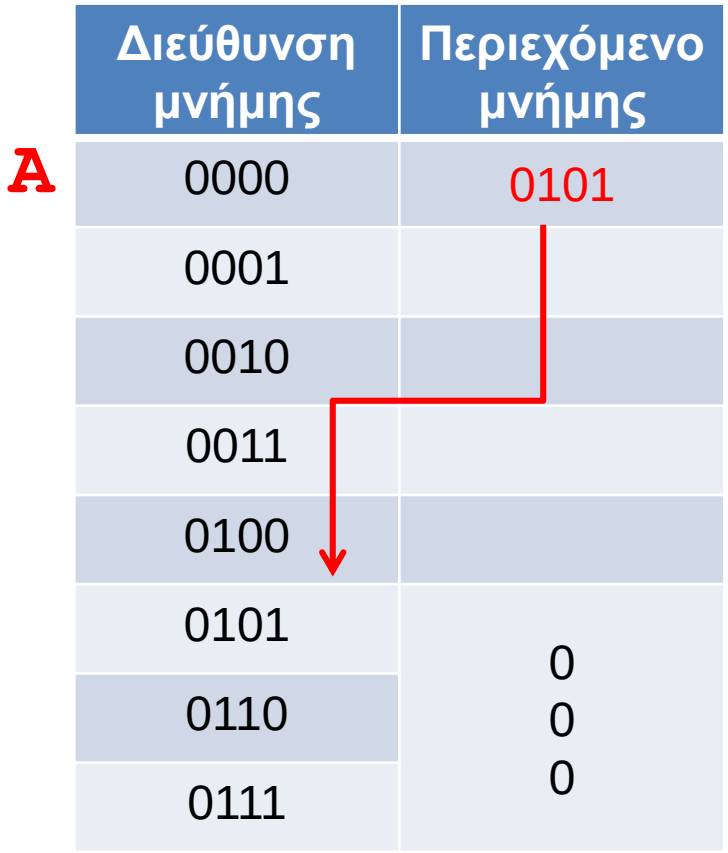
Με την εντολή new δεσμεύουμε δύο θέσεις ακεραίων και η αναφορά του A δείχνει σε αυτό το χώρο που δεσμεύσαμε



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)



Αντικείμενα κλάσεων

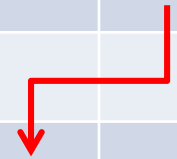
- Τι γίνεται με τα αντικείμενα κλάσεων που ορίσαμε εμείς?
- Παράδειγμα: Η κλάση Person (ToyClass από το βιβλίο).

```
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

```
Person varP = new Person("Bob", 1);
```

varP



Διεύθυνση μνήμης	Περιεχόμενο μνήμης
0000	0010
0001	
0010	"Bob"
0011	
0100	
0101	1
0110	
0111	

Αναθέσεις μεταξύ αντικειμένων

Τι θα τυπώσει το παρακάτω πρόγραμμα?

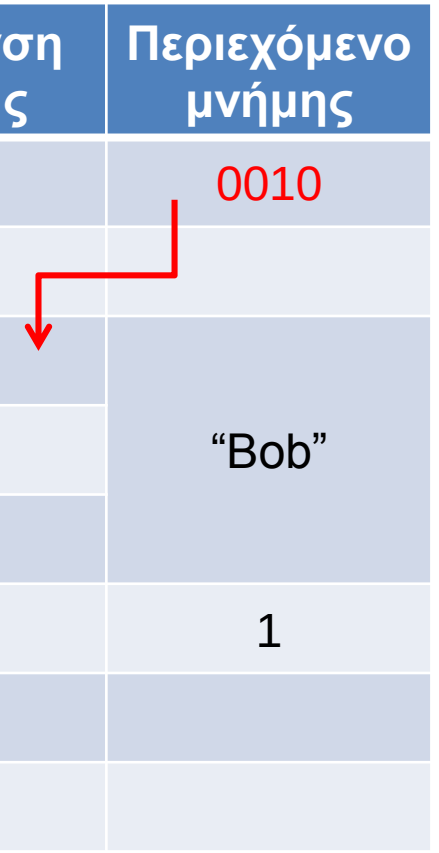
```
Person varP1 = new Person("Bob", 1);
Person varP2;
varP2 = varP1;
varP2.set("Ann",2);
System.out.println(varP1);
```

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

Αναθέσεις μεταξύ αντικειμένων

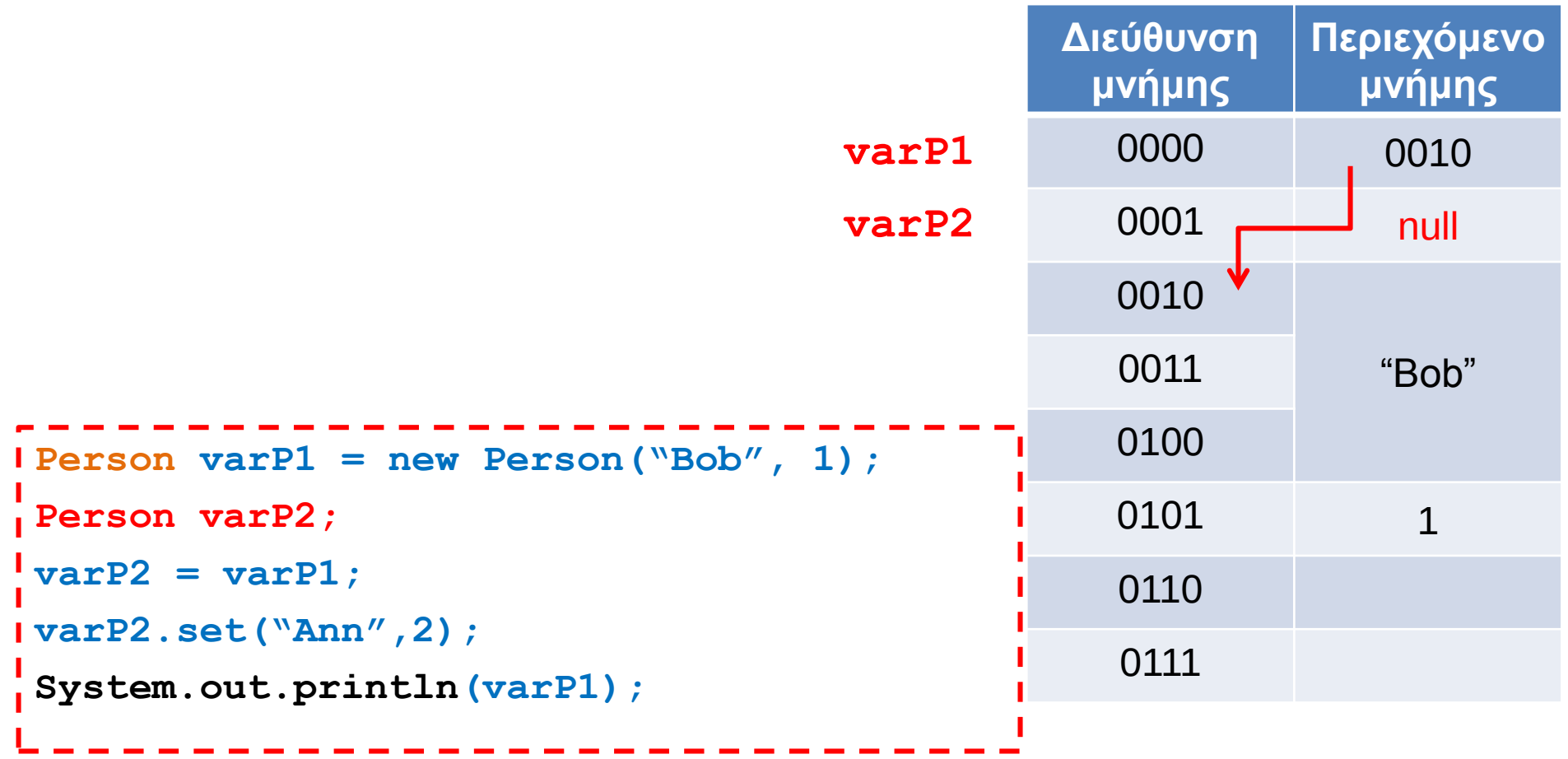
```
Person varP1 = new Person("Bob", 1);
Person varP2;
varP2 = varP1;
varP2.set("Ann",2);
System.out.println(varP1);
```

varP1



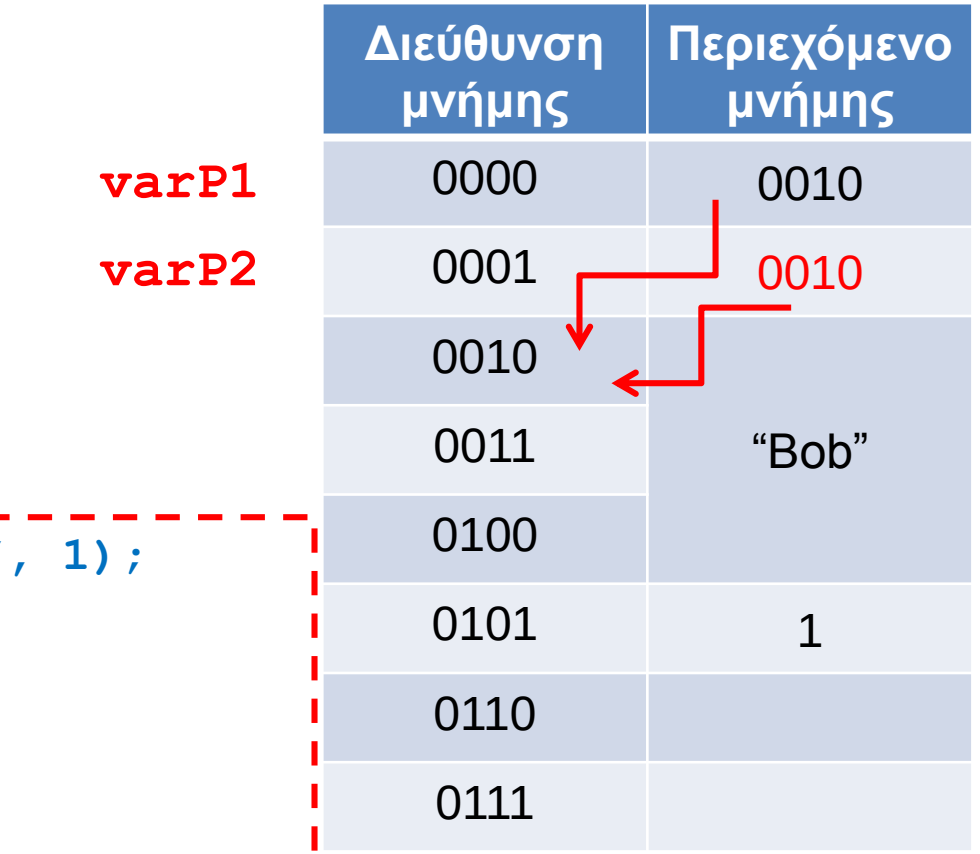
Αναθέσεις μεταξύ αντικειμένων

```
Person varP1 = new Person("Bob", 1);
Person varP2;
varP2 = varP1;
varP2.set("Ann",2);
System.out.println(varP1);
```



Αναθέσεις μεταξύ αντικειμένων

```
Person varP1 = new Person("Bob", 1);
Person varP2;
varP2 = varP1;
varP2.set("Ann",2);
System.out.println(varP1);
```



Αναθέσεις μεταξύ αντικειμένων

Η αλλαγή θα γίνει στο χώρο μνήμης που δείχνει ο **varP2**
Αυτός είναι ο ίδιος όπως αυτός που δείχνει και ο **varP1**

varP1

varP2

Διεύθυνση μνήμης	Περιεχόμενο μνήμης
0000	0010
0001	0010
0010	
0011	"Ann"
0100	
0101	2
0110	
0111	

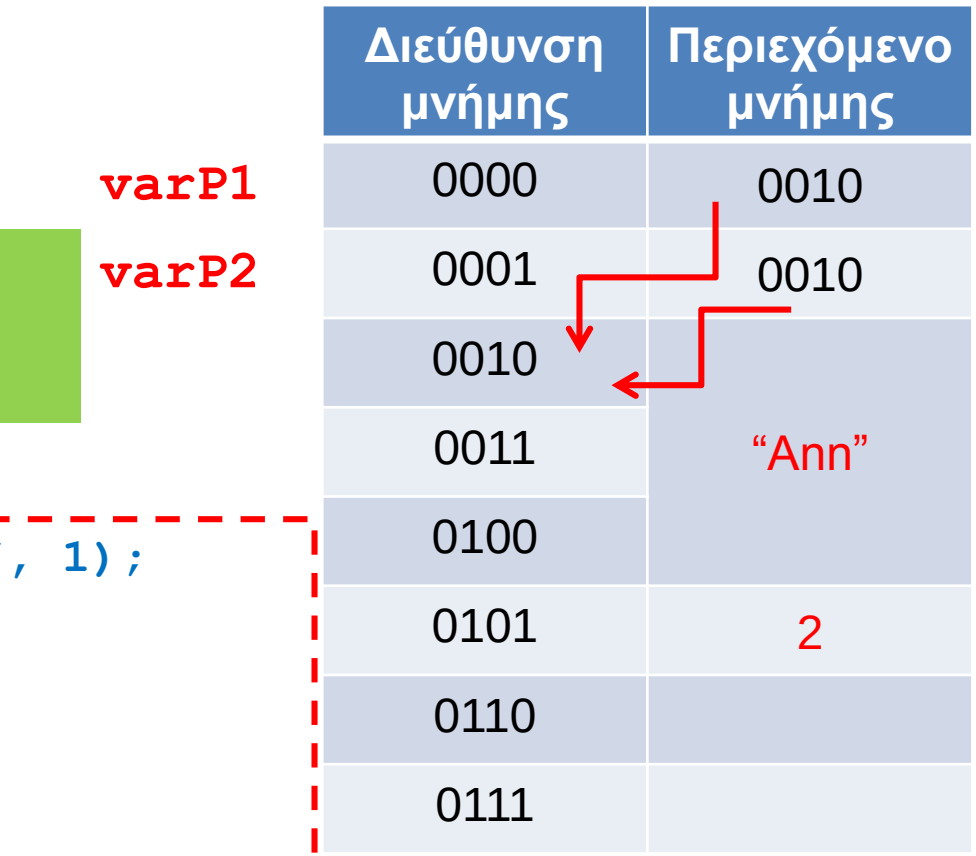
```
Person varP1 = new Person("Bob", 1);
Person varP2;
varP2 = varP1;
varP2.set("Ann",2);
System.out.println(varP1);
```

Αναθέσεις μεταξύ αντικειμένων

Τυπώνει “Ann 2”

Αλλάζοντας τα περιεχόμενα της θέσης μνήμης στην οποια δείχνει ο **varP2** αλλάζουμε και το **varP1**

```
Person varP1 = new Person(“Bob”, 1);
Person varP2;
varP2 = varP1;
varP2.set(“Ann”,2);
System.out.println(varP1);
```



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.

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.

Παράδειγμα

```
public class ClassParameterDemo
{
     public static void main(String[] args)
     {
          Person aPerson = new Person("Mr. White", 1);
          System.out.println(aPerson);
          Person anotherperson = new Person("Heisnberg", 2);
          System.out.println(
                    "Now we call copier with aPerson as argument.");
          anotherPerson.copier(aPerson);
          System.out.println(aPerson);
      }
}
```

Τι θα τυπώσει?

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

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

}
```

Heisenberg 2

Εξήγηση

```
Person aPerson = new
    Person("Mr. White", 1);
Person anotherPerson = new
    Person("Heisenberg", 2);
```

Διεύθυνση μνήμης	Περιεχόμενο μνήμης
0010 (aPerson)	0200
0011 (anotherPerson)	0300
0100	
…	
0200	"Mr. White" 1
0300	"Heisenberg" 2
0110	
0111	

Εξήγηση

aPerson

anotherPerson

other

```
anotherPerson.copier(aPerson);
```

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

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

}
```

other = aPerson

Διεύθυνση μνήμης	Περιεχόμενο μνήμης
0010	0200
0011	0300
0100	0200
…	
0200	“Mr. White” 1
0300	“Heisenberg” 2
0110	
0111	

Εξήγηση

aPerson

anotherPerson

other

```
anotherPerson.copier(aPerson);
```

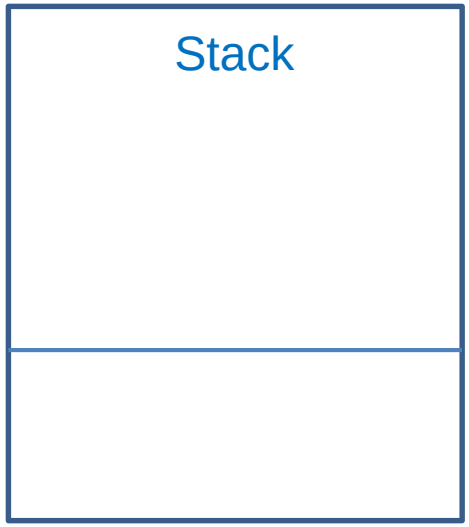
```
public class Person
{
    private String name;
    private int number;

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

}
```

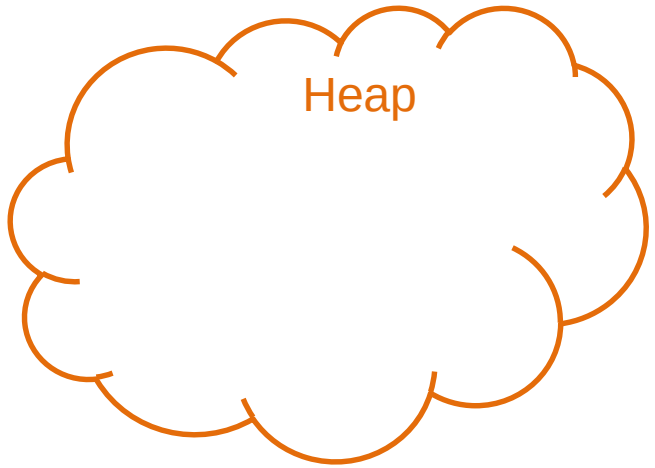
Διεύθυνση μνήμης	Περιεχόμενο μνήμης
0010	0200
0011	0300
0100	0200
…	
0200	“Heisenberg” 2
0300	“Heisenberg” 2
0110	
0111	

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



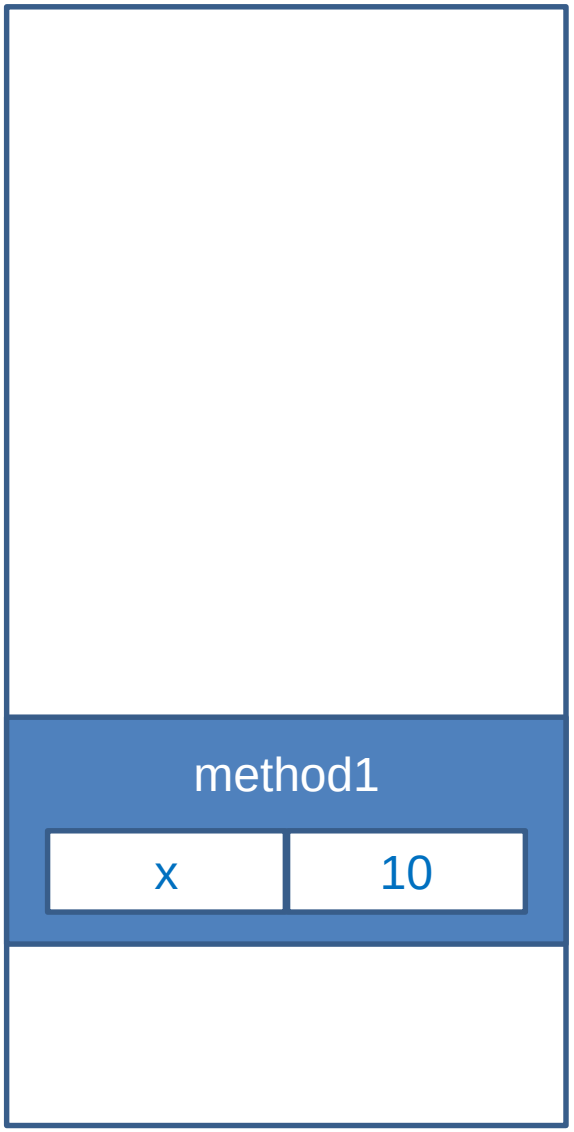


Stack

- 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.

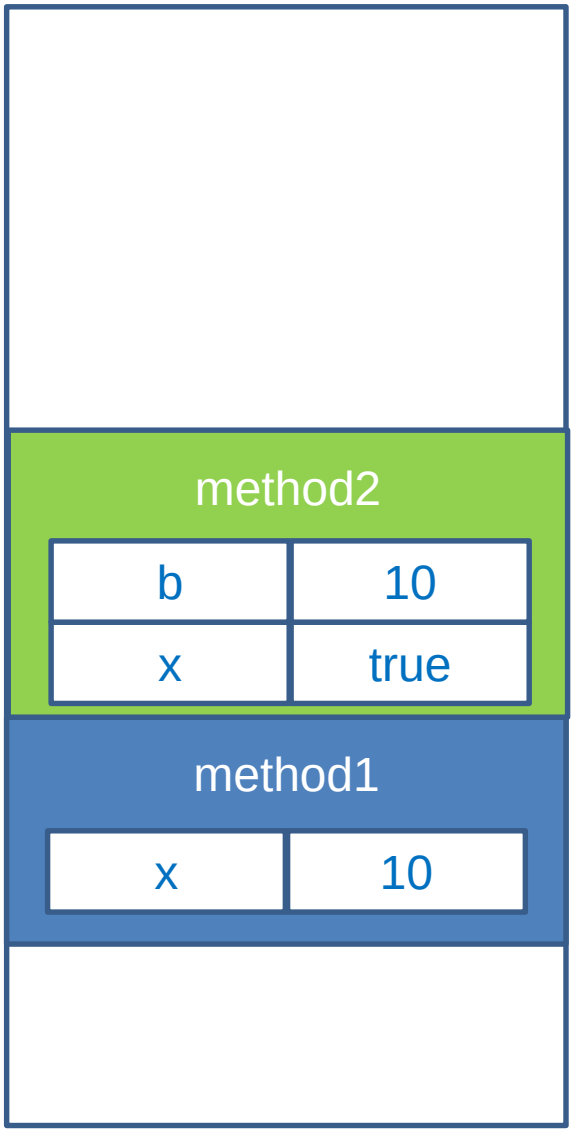
Παράδειγμα

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



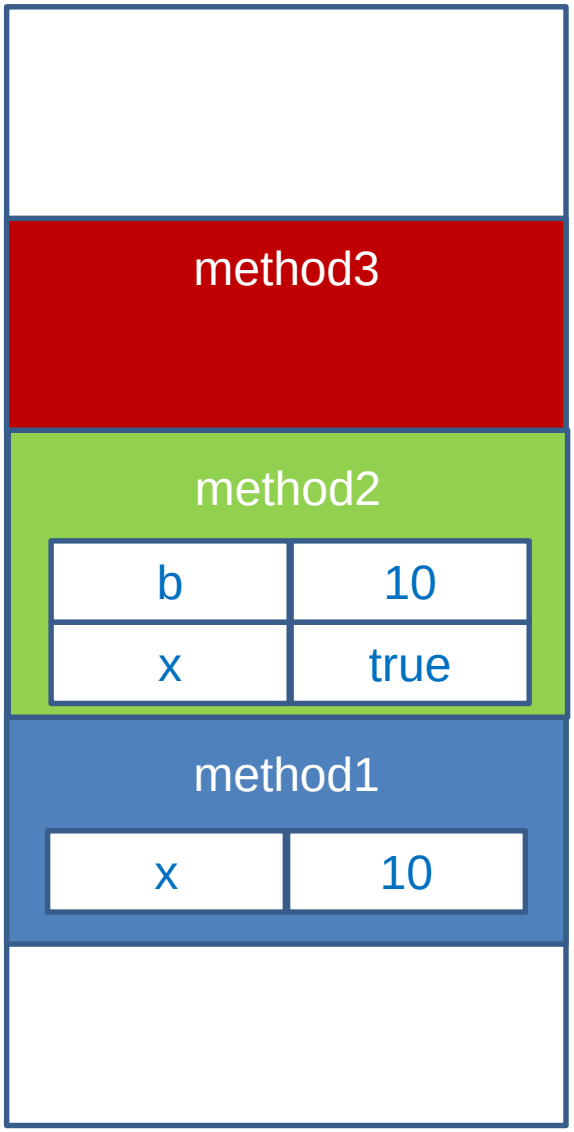
Παράδειγμα

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

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



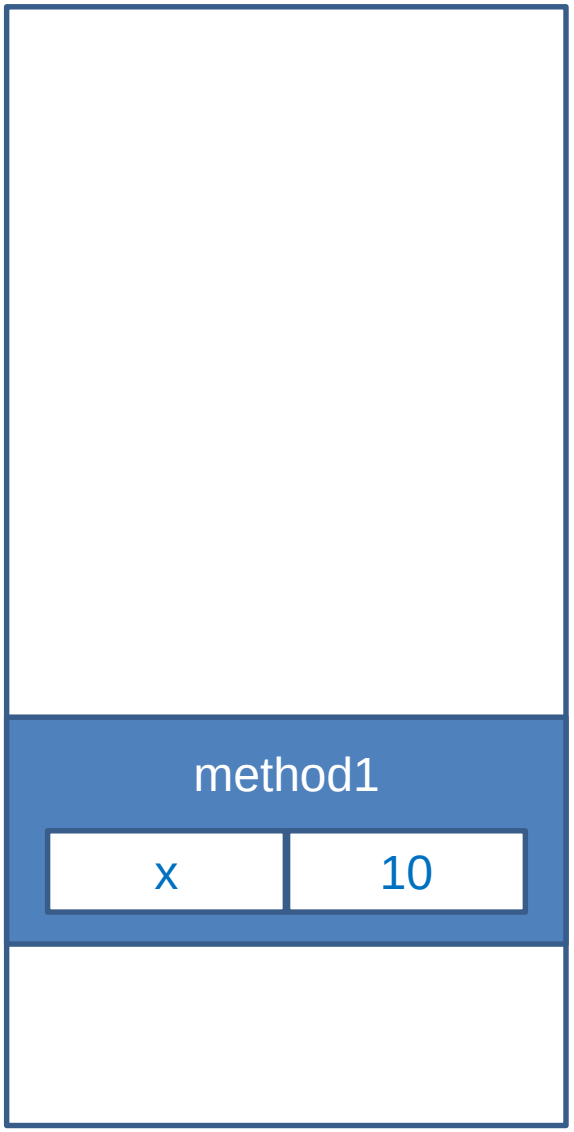
Παράδειγμα

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

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

public void method3()
{…}
```



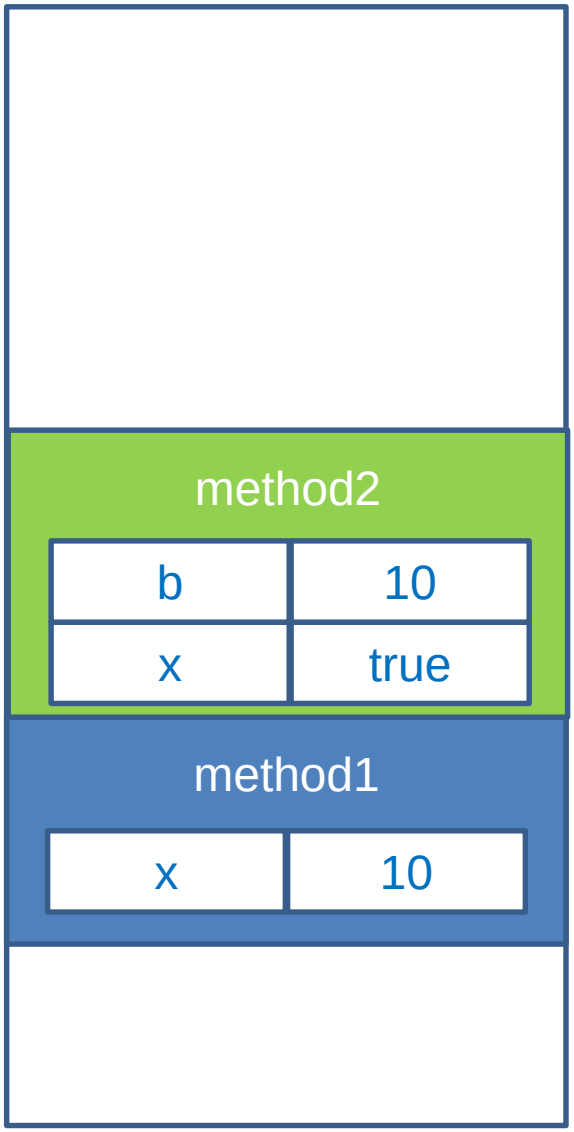
Παράδειγμα

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



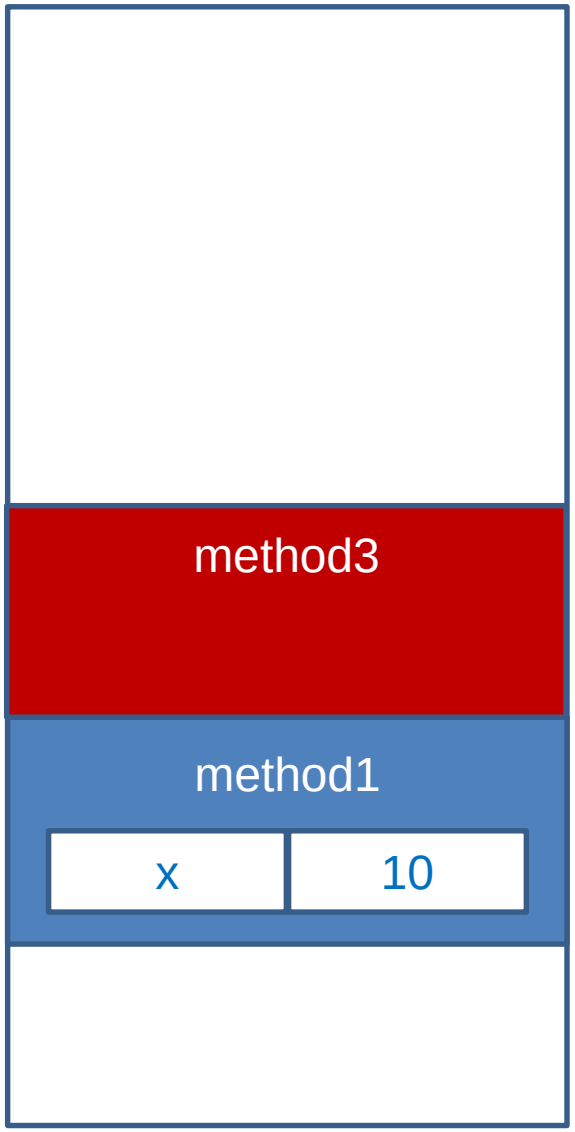
Παράδειγμα

```
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()
{…}
```

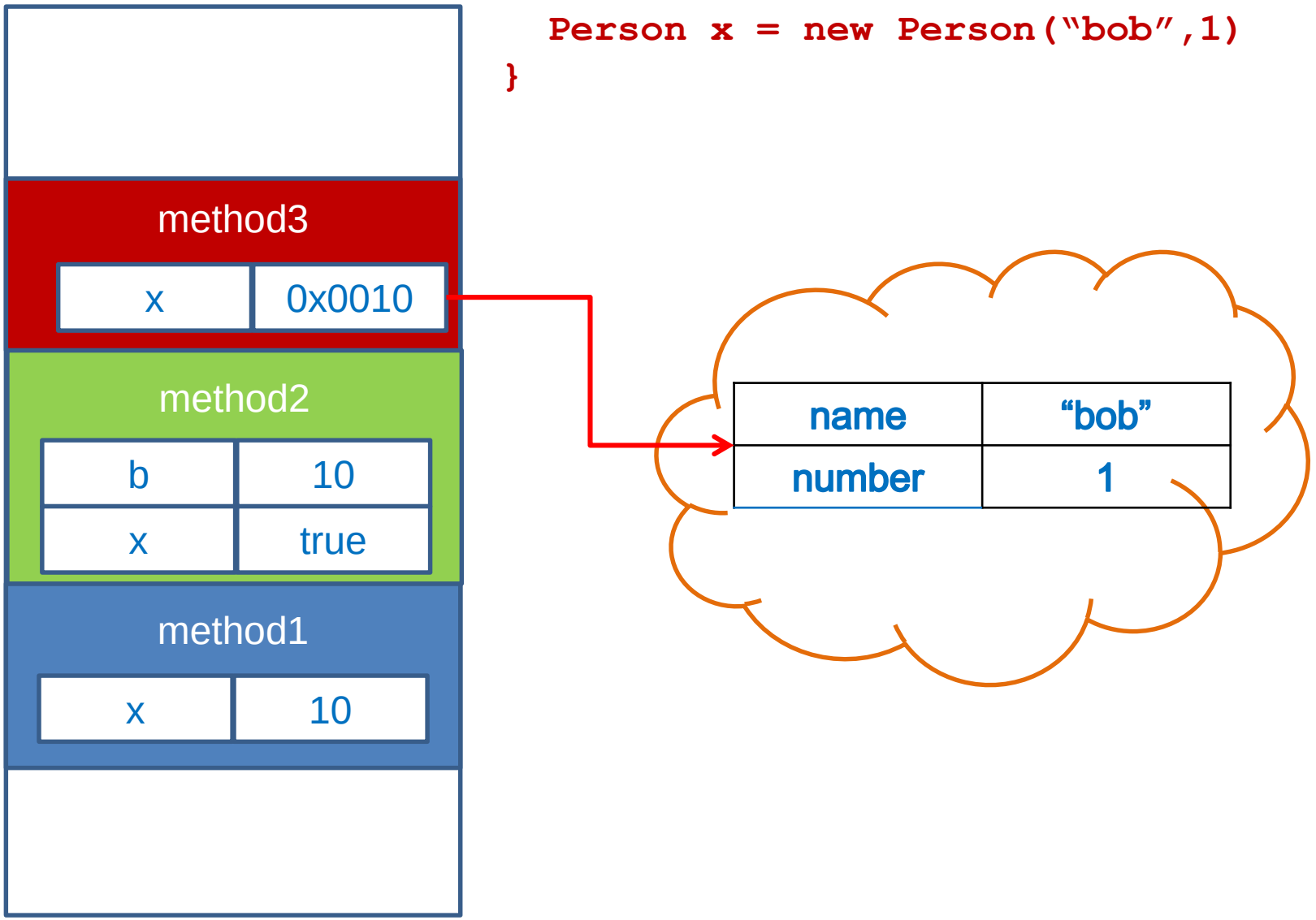


Heap

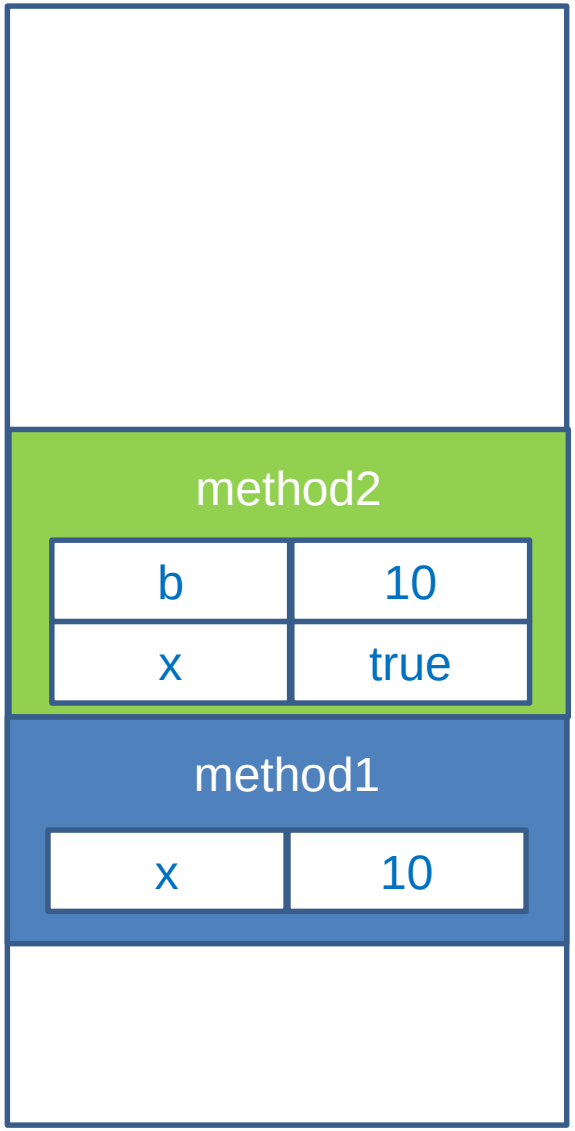
- Όταν μέσα σε μία μέθοδο δημιουργούμε ένα αντικείμενο με την new γίνονται τα εξής
  - στο πλαίσιο (frame) της μεθόδου (στη στοίβα) υπάρχει μια τοπική μεταβλητή που κρατάει την αναφορά στο αντικείμενο
  - Η κλήση της new δεσμεύει χώρο μνήμης στο σωρό (heap) για να κρατήσει τα πεδία του αντικειμένου.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

Παράδειγμα

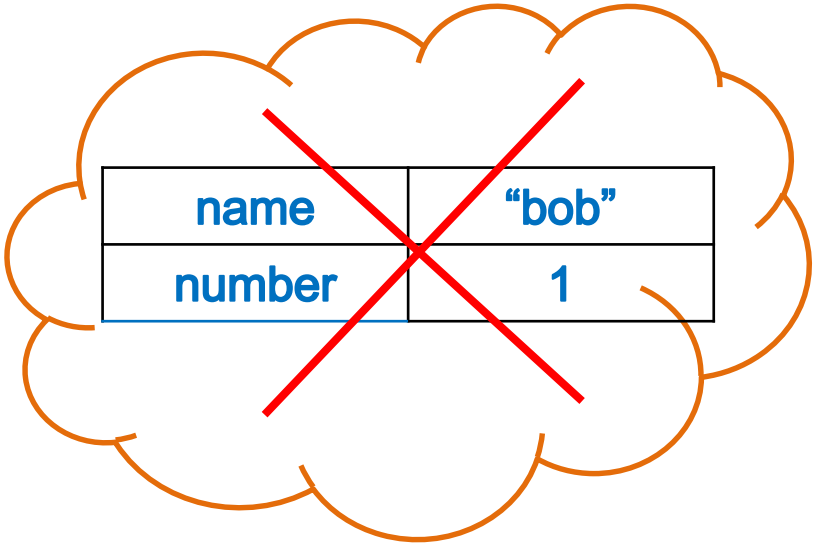
```
public void method3()
{
   Person x = new Person("bob",1)
}
```



Παράδειγμα



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

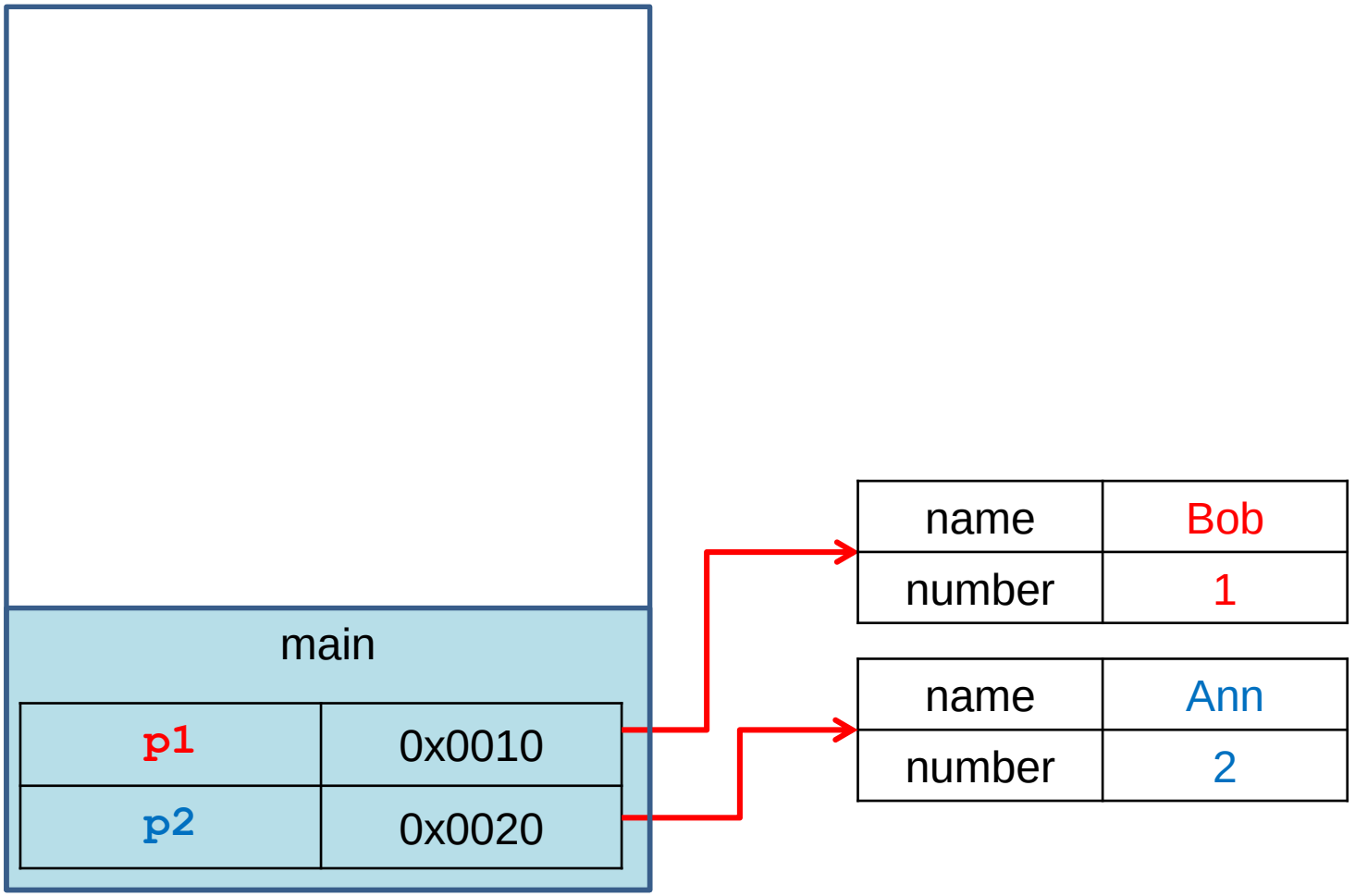


Αν δεν υπάρχουν άλλες αναφορές στο αντικείμενο τότε ο garbage collector αποδεσμεύει τη μνήμη του αντικειμένου

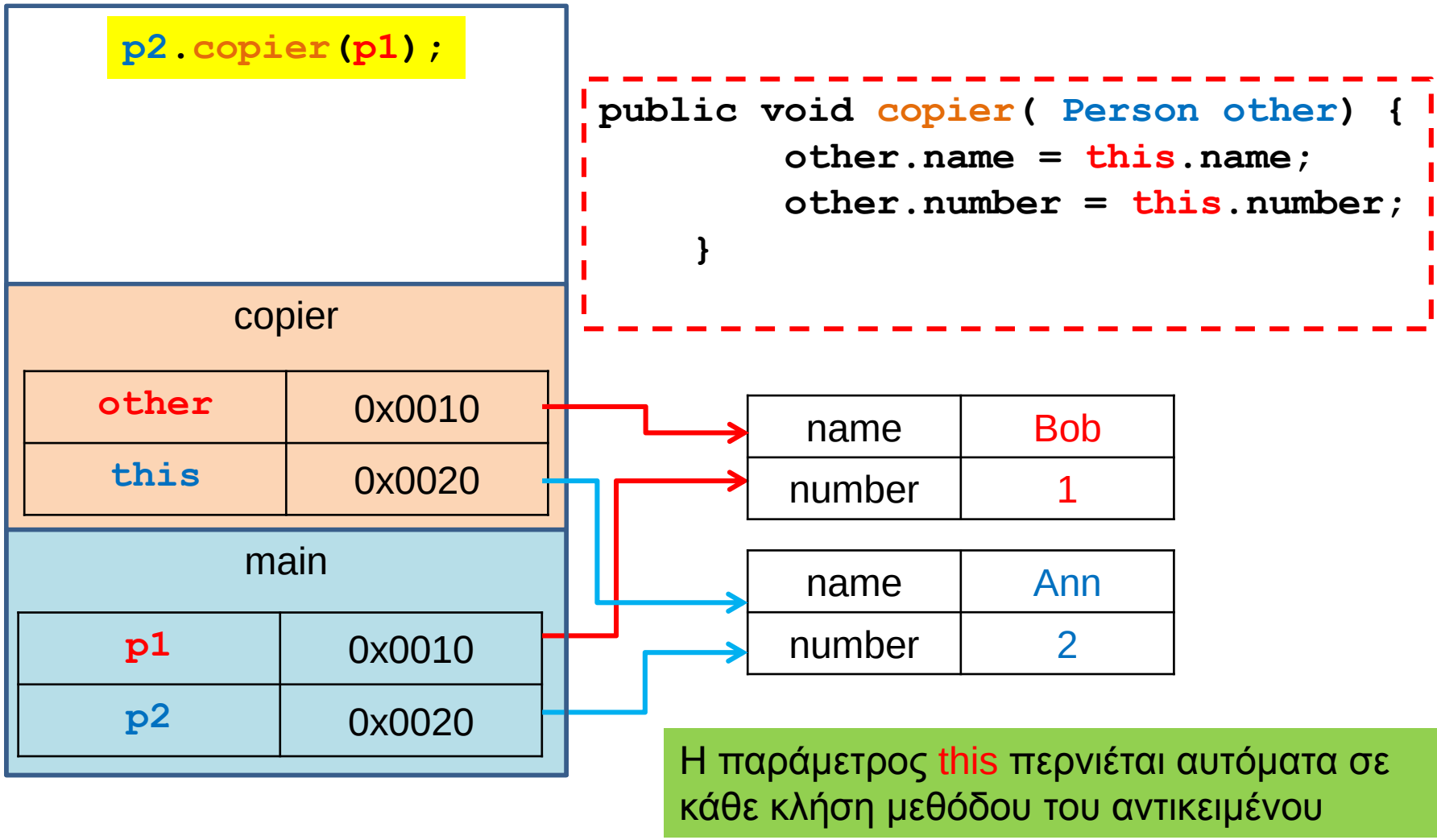
Παράδειγμα

```
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);
     }
}
```

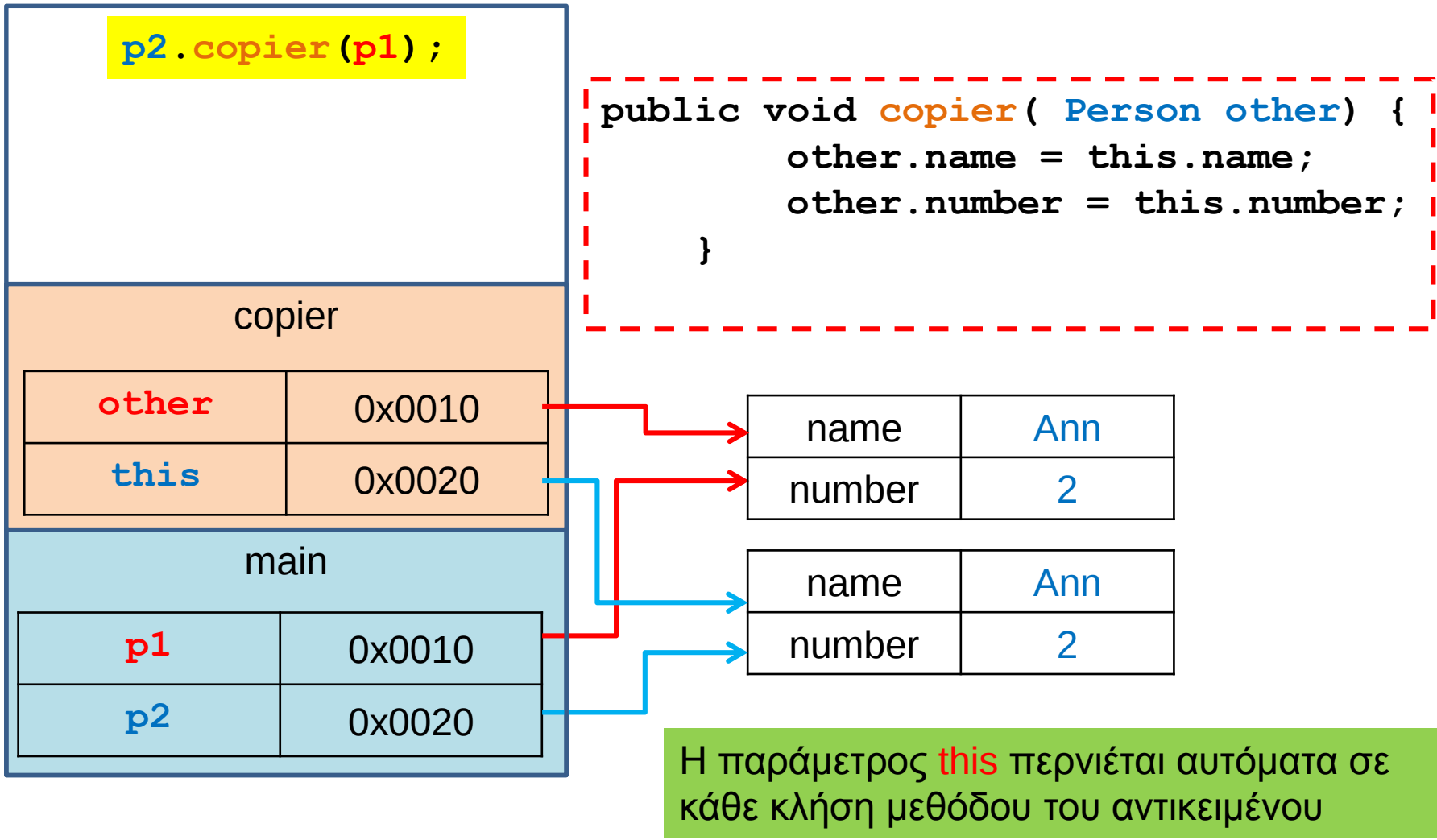
Εξέλιξη του προγράμματος



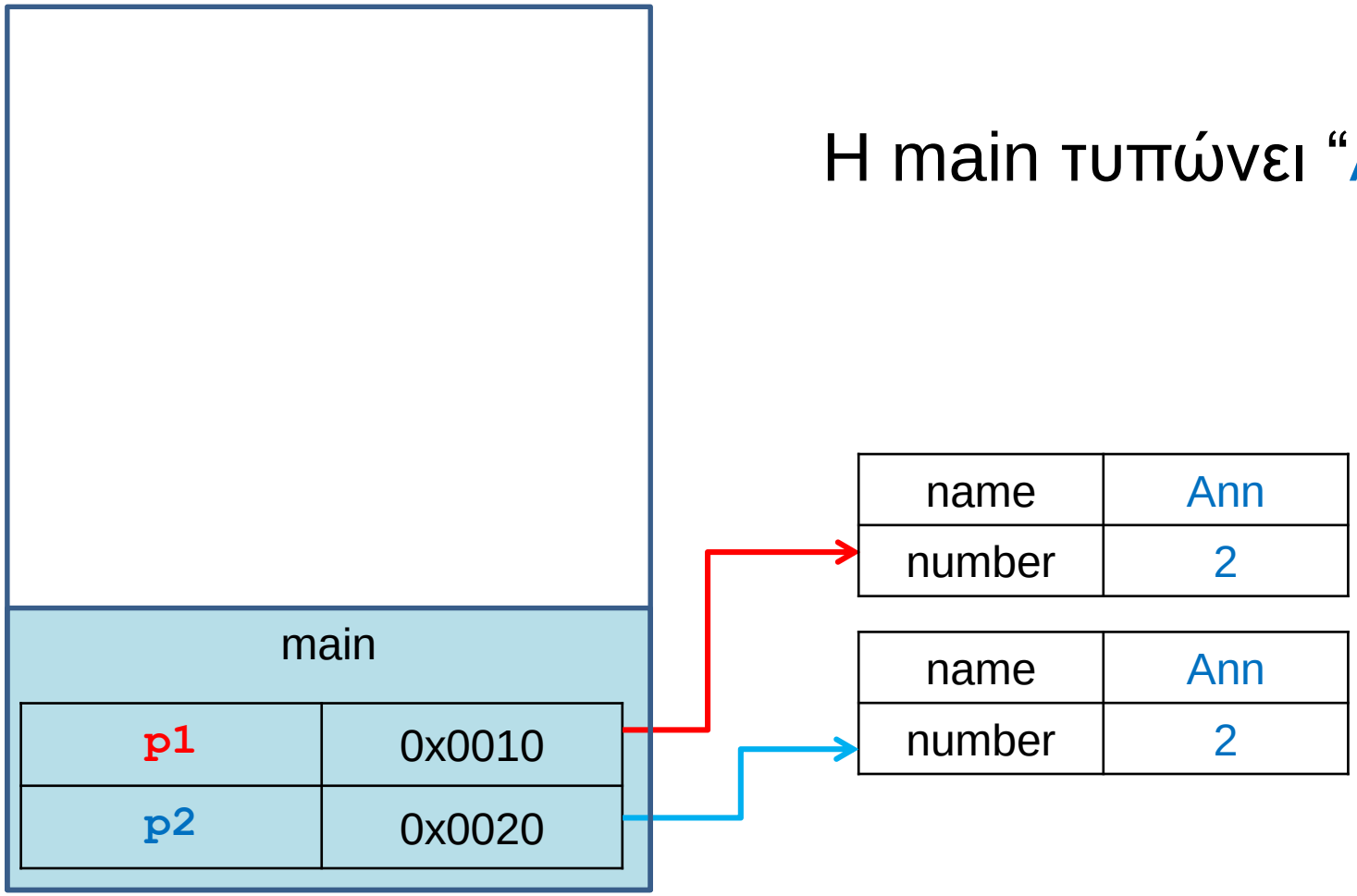
Εξέλιξη του προγράμματος



Εξέλιξη του προγράμματος



Εξέλιξη του προγράμματος



Η main τυπώνει “Ann 2”

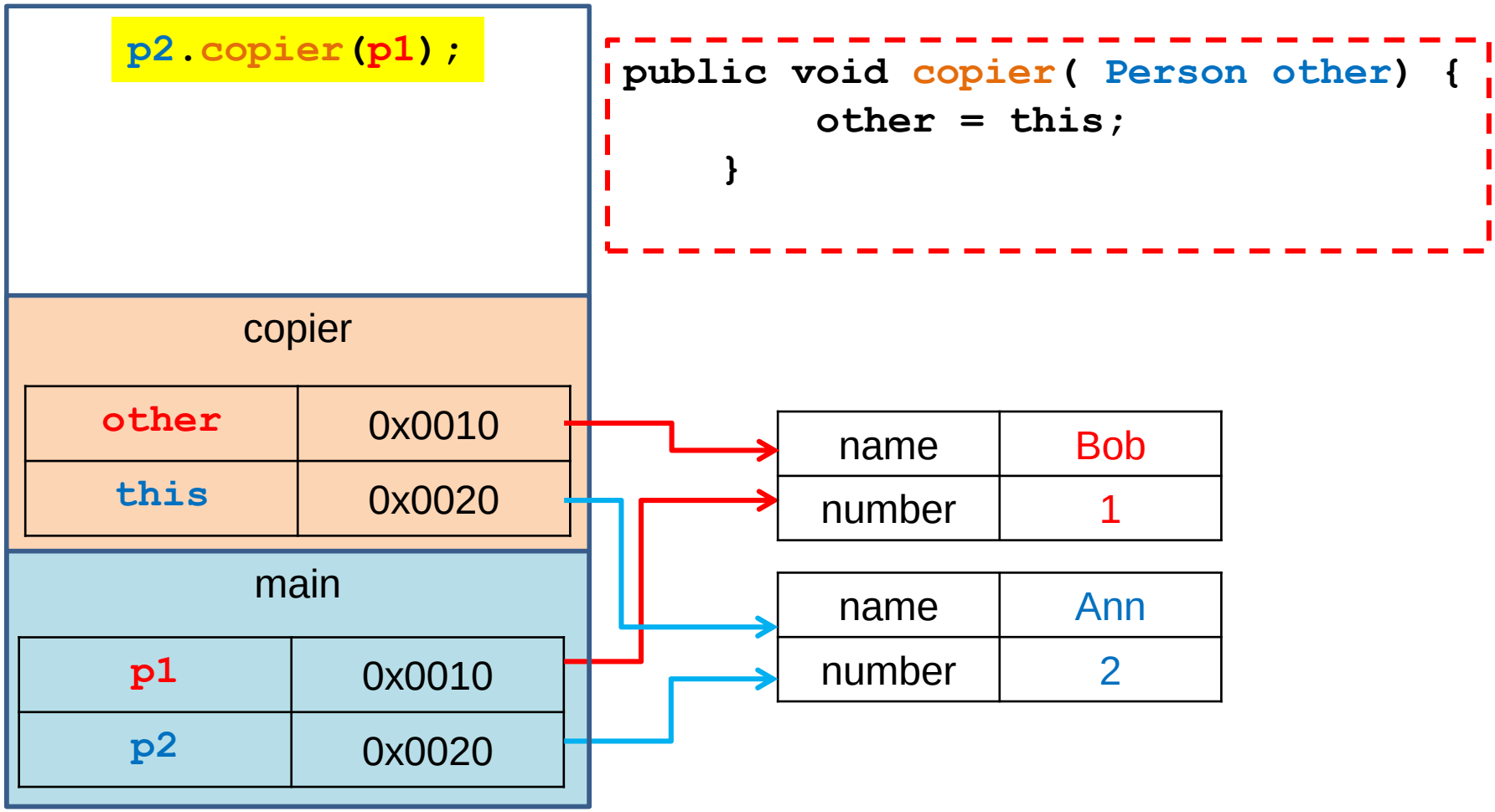
Μια άλλη υλοποίηση της copier

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

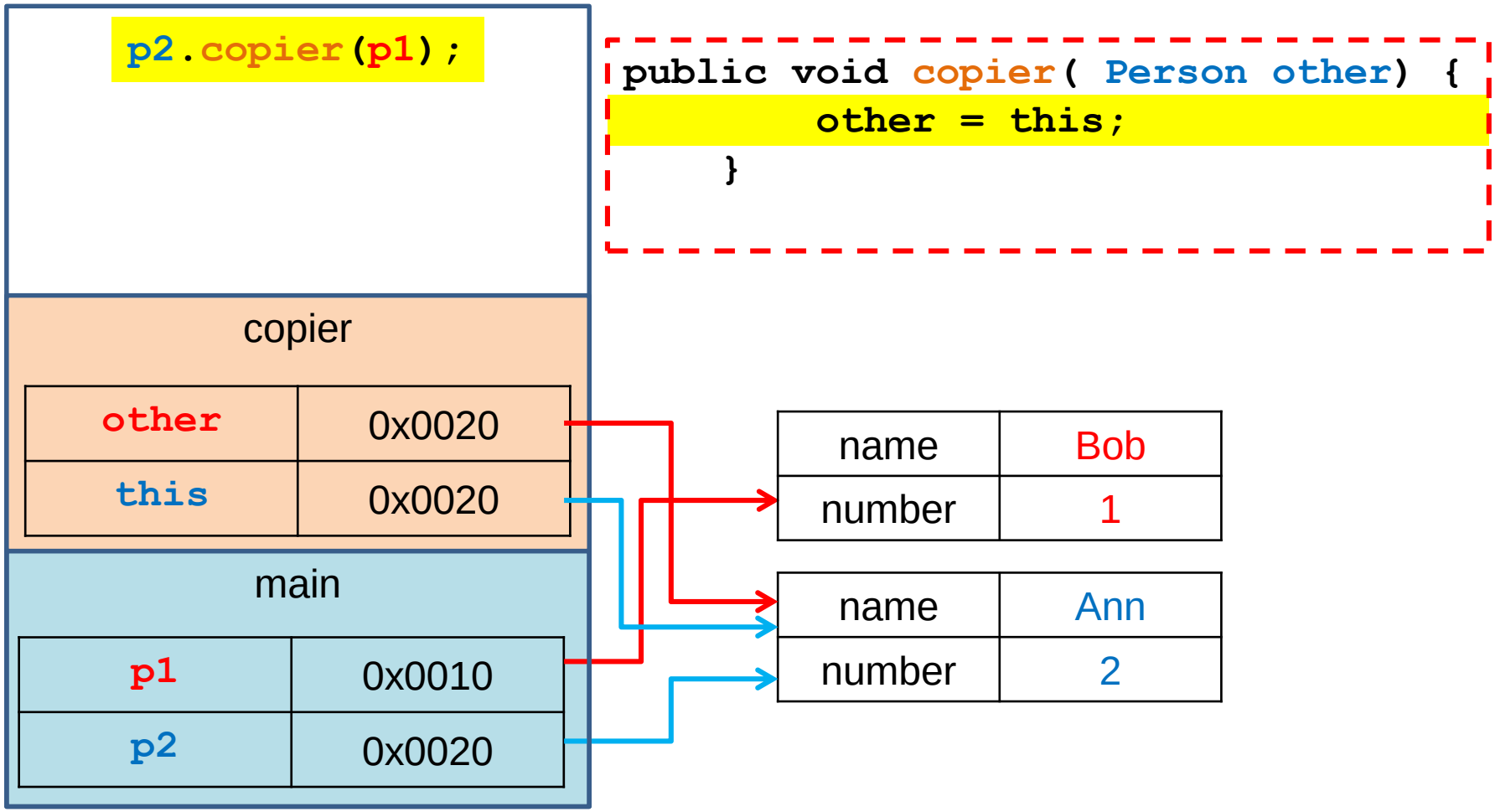
```
public class ClassParameterDemo
{
    public static void main(String[] args)
    {
        Person p1 = new Person("Bob", 1);
        Person p2 = new Person("Ann", 2);
        p2.copier(p1);
        System.out.println(p1);
     }
}
```

Τι θα τυπώσει?

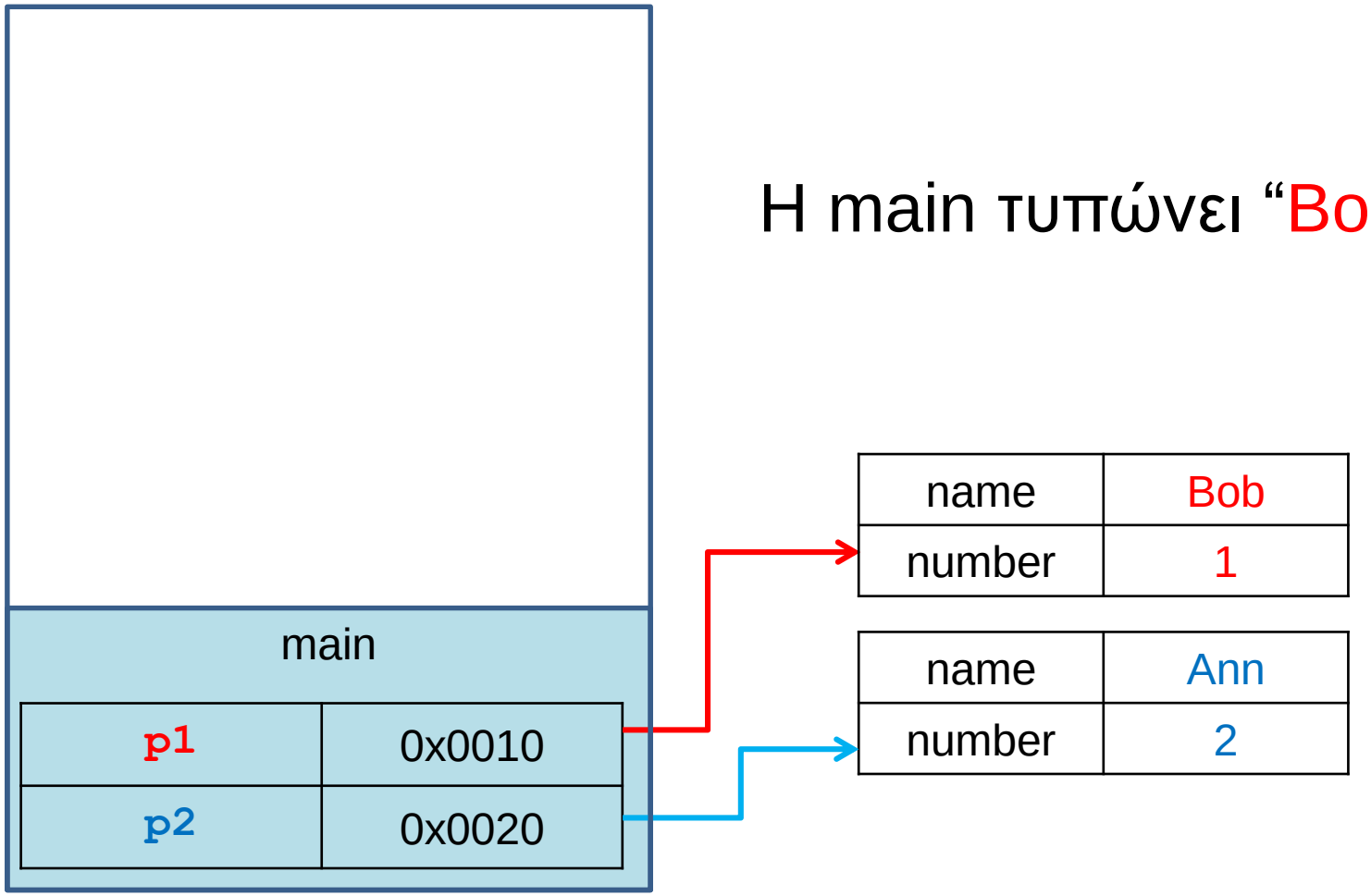
Εξέλιξη του προγράμματος



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);
    }
```

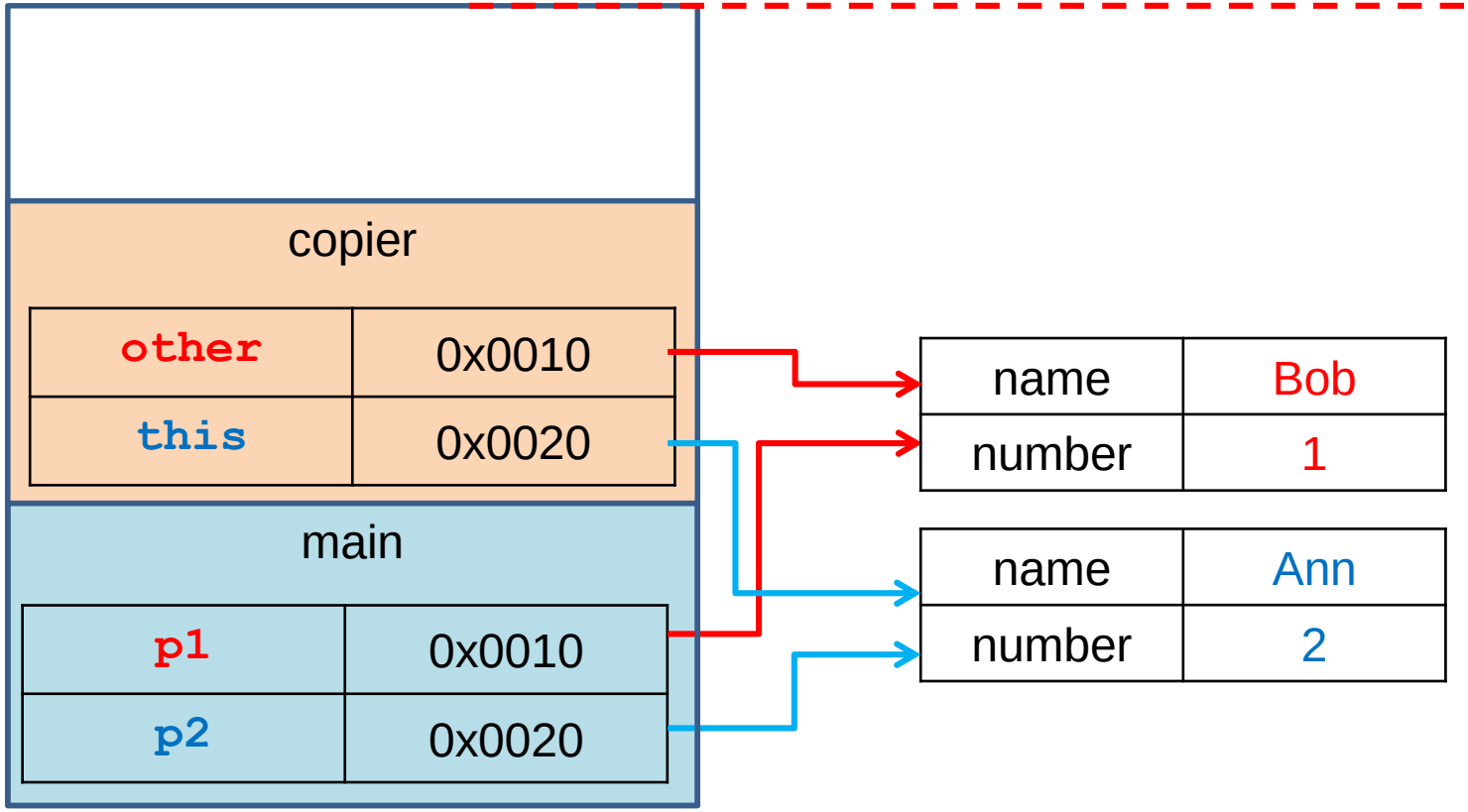
```
public class ClassParameterDemo
{
    public static void main(String[] args)
    {
        Person p1 = new Person("Bob", 1);
        Person p2 = new Person("Ann", 2);
        p2.copier(p1);
        System.out.println(p1);
     }
}
```

Τι θα τυπώσει?

Εξέλιξη του προγράμματος

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

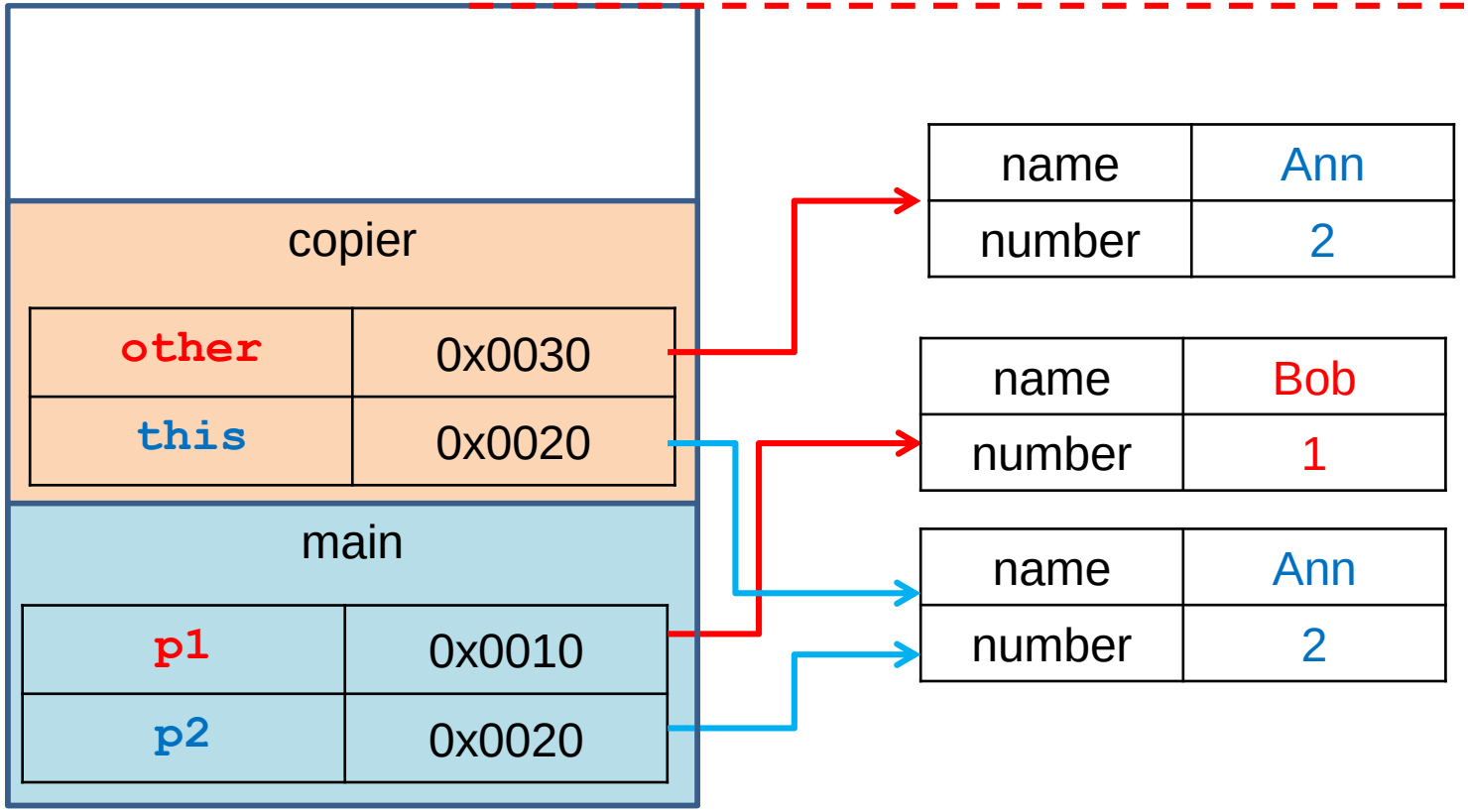
```
public void copier( Person other) {
  other = new Person(this.name, this.number);
}
```



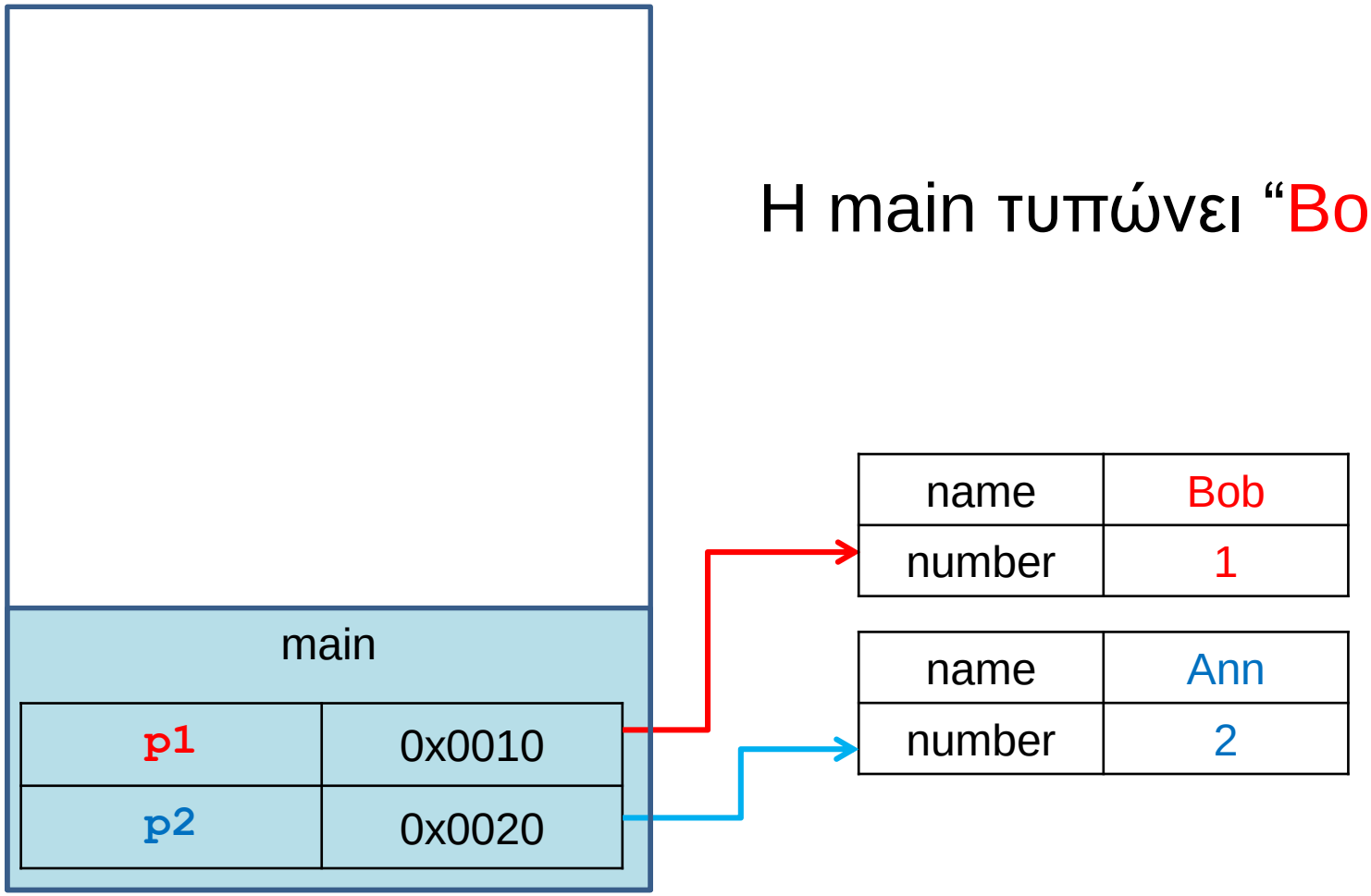
Εξέλιξη του προγράμματος

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

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



Εξέλιξη του προγράμματος



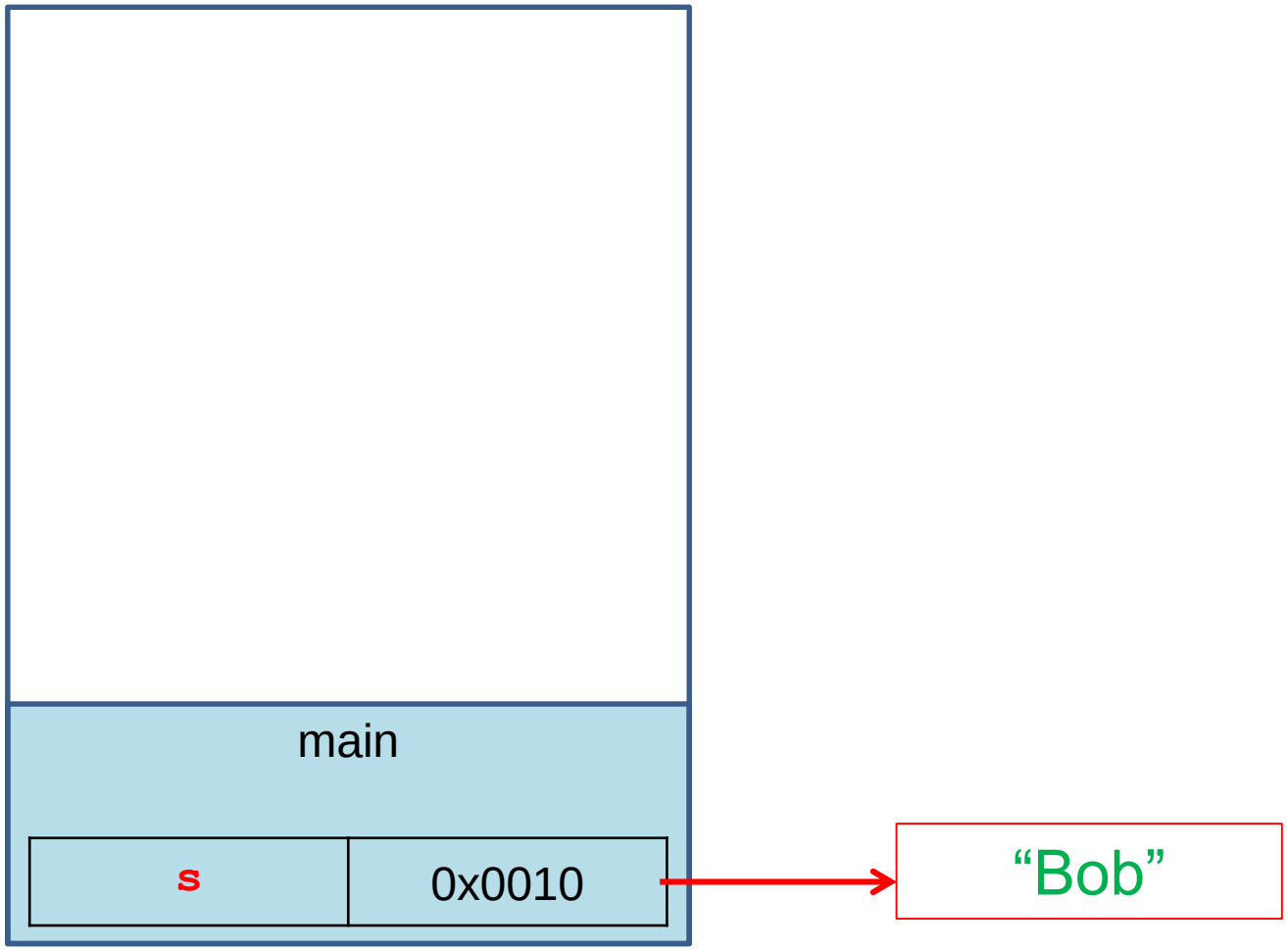
Η main τυπώνει “Bob 1”

Άλλο ένα παράδειγμα

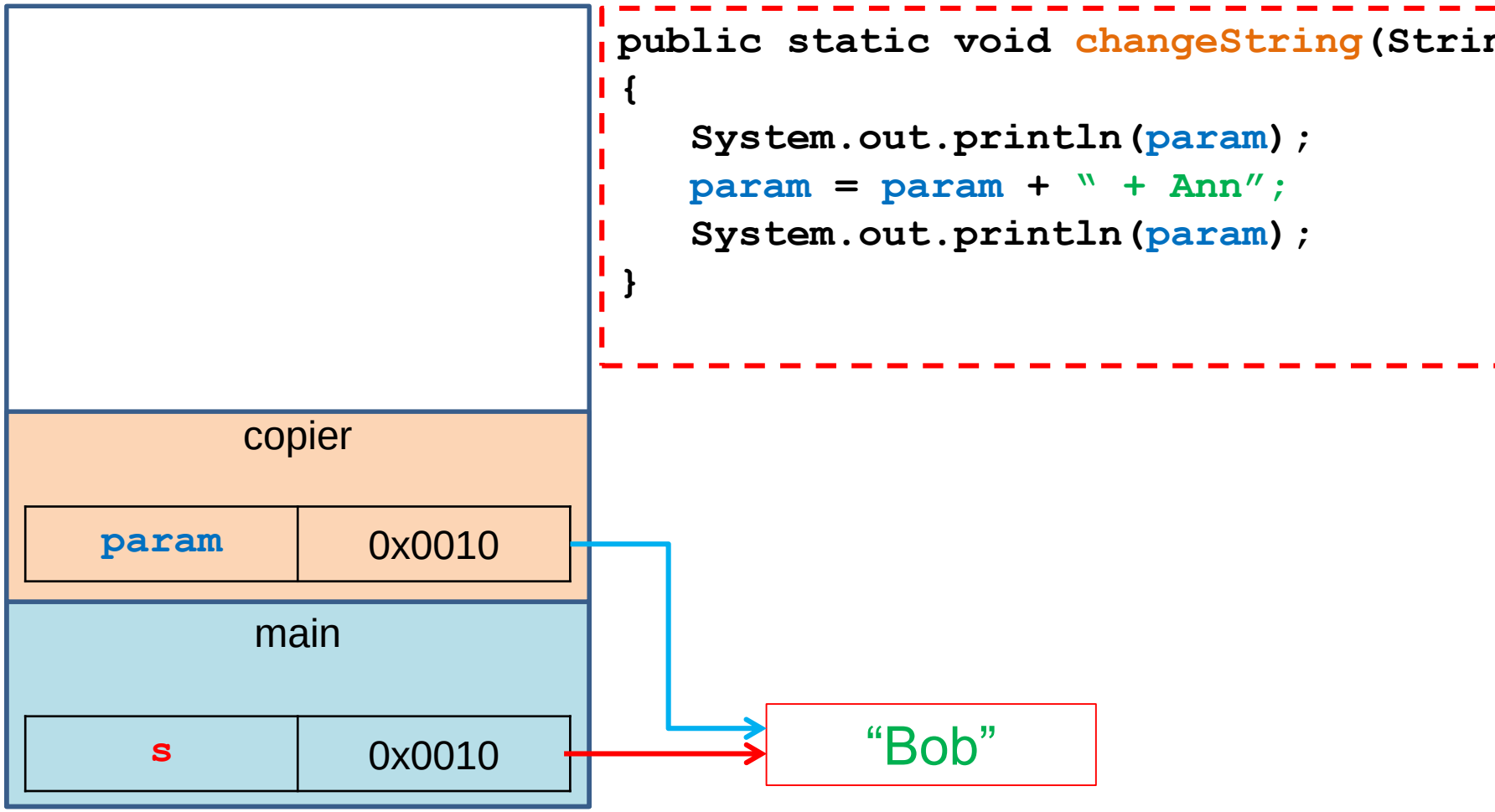
```
public class StringParameterDemo
{
    public static void main(String[] args)
    {
        String s = "Bob";
        changeString(s);
        System.out.println(s);
     }

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

Τι θα τυπώσει?

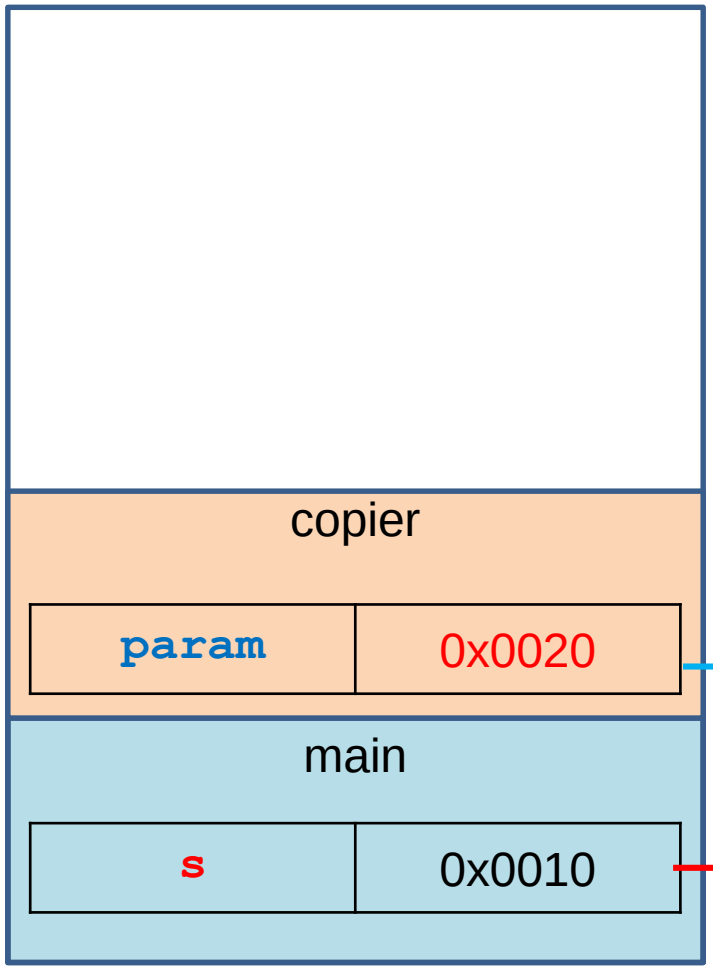
Εξέλιξη του προγράμματος



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