ΤΕΧΝΙΚΕΣ ΑΝΤΙΚΕΙΜΕΝΟΣΤΡΑΦΟΥΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ

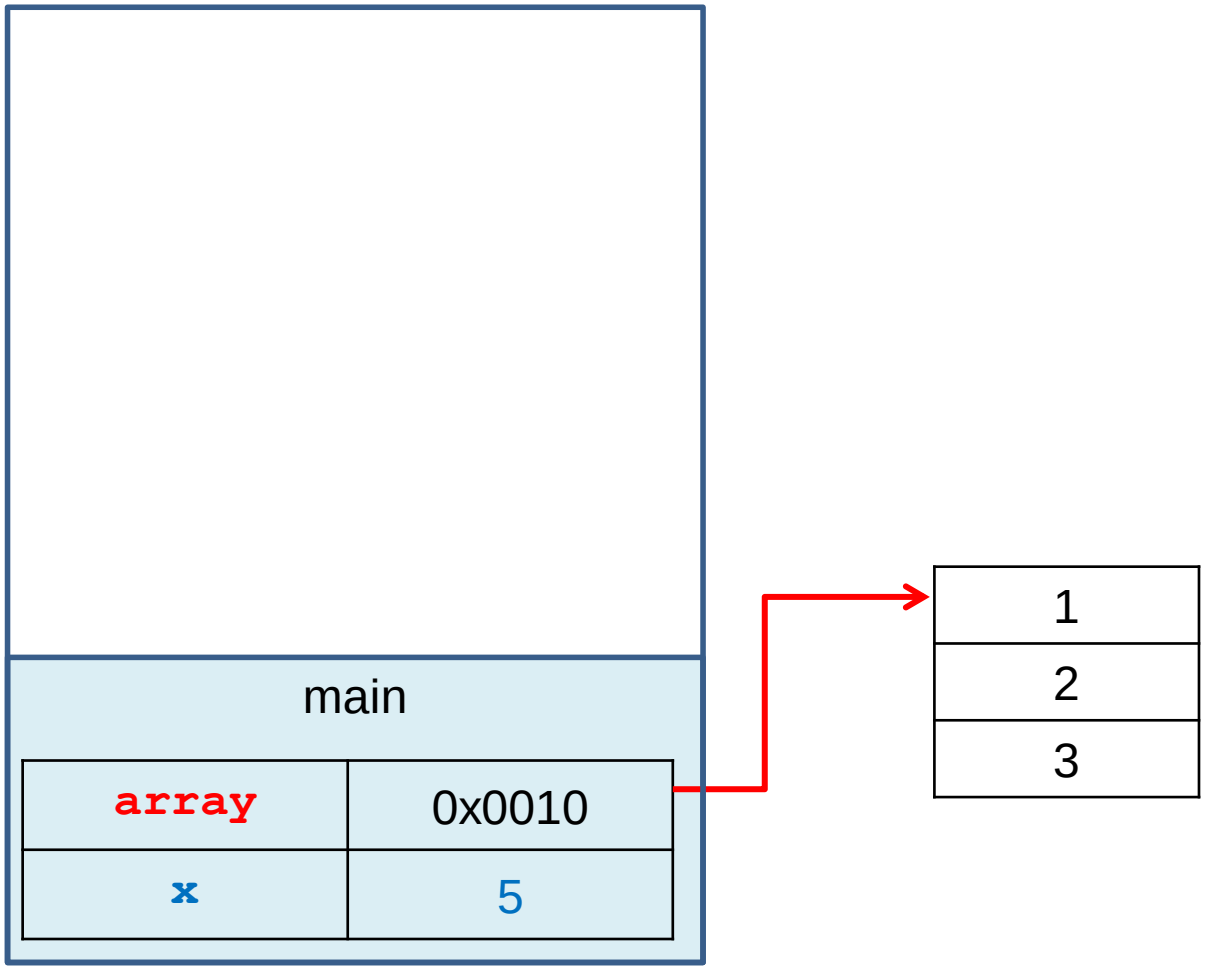
Copy Constructor

Deep and Shallow Copies

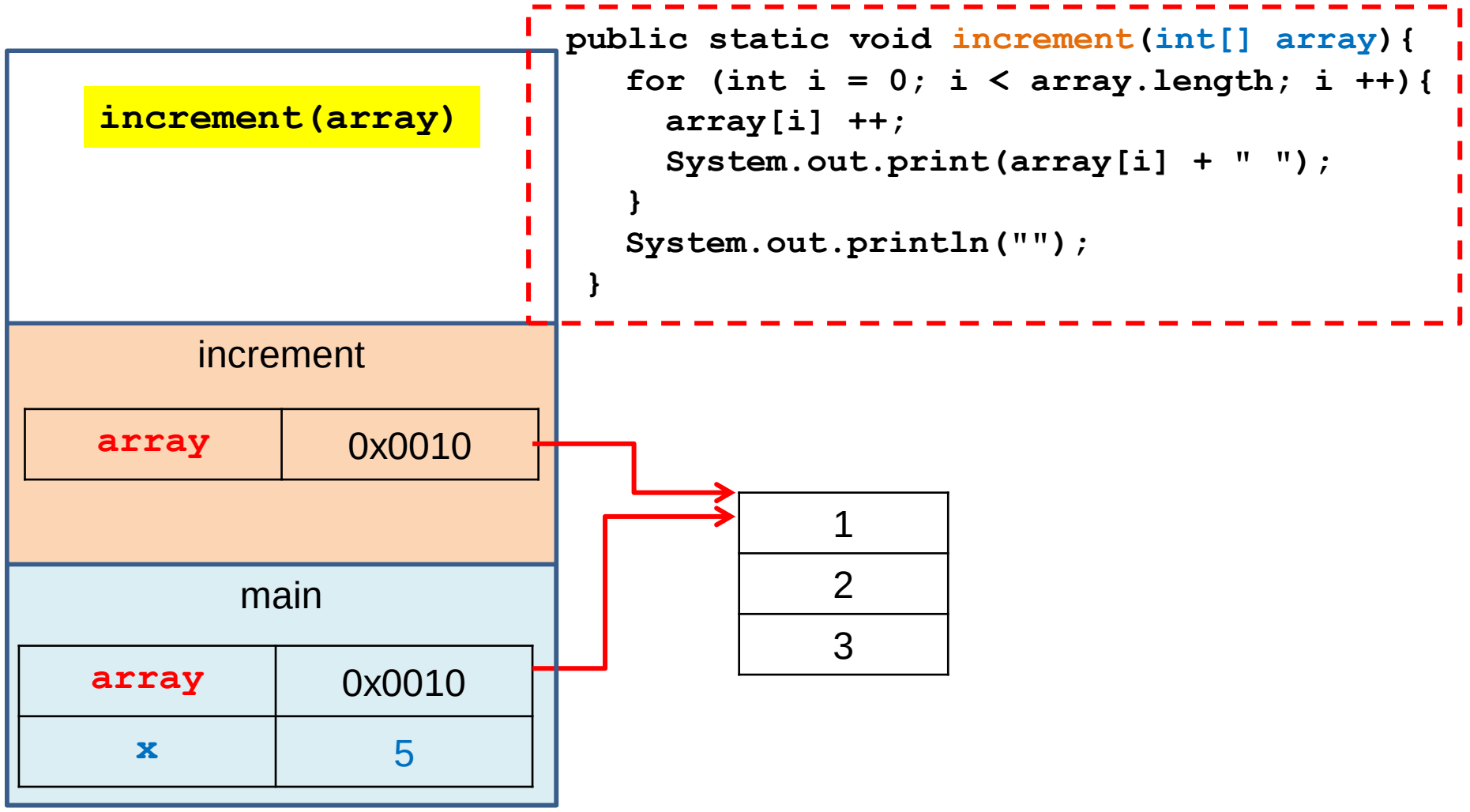
```
class ArrayVar
{
  public static void main(String[] args){
    int[] array = {1,2,3};
    int x = 4;

    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

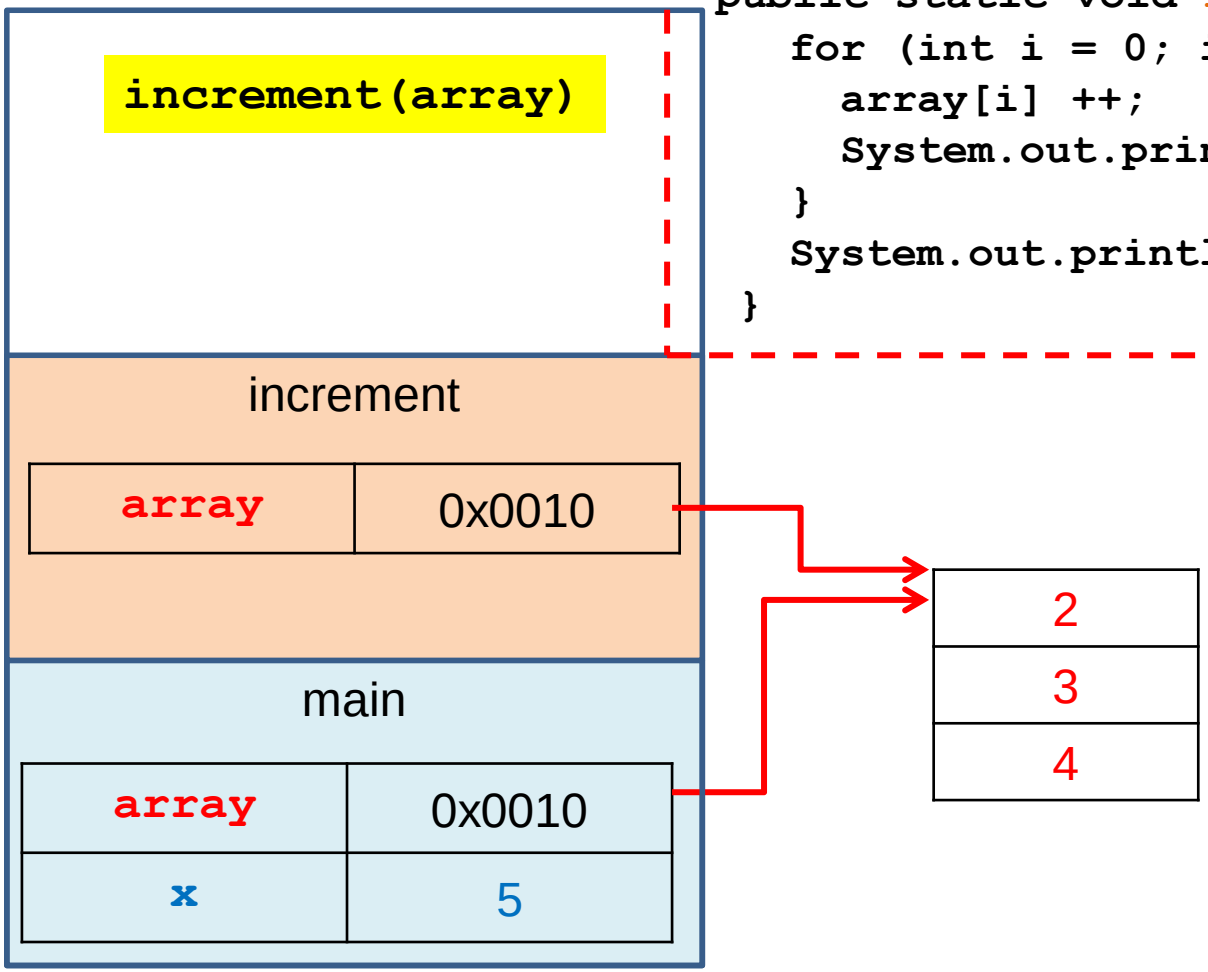


Πέρασμα παραμέτρων

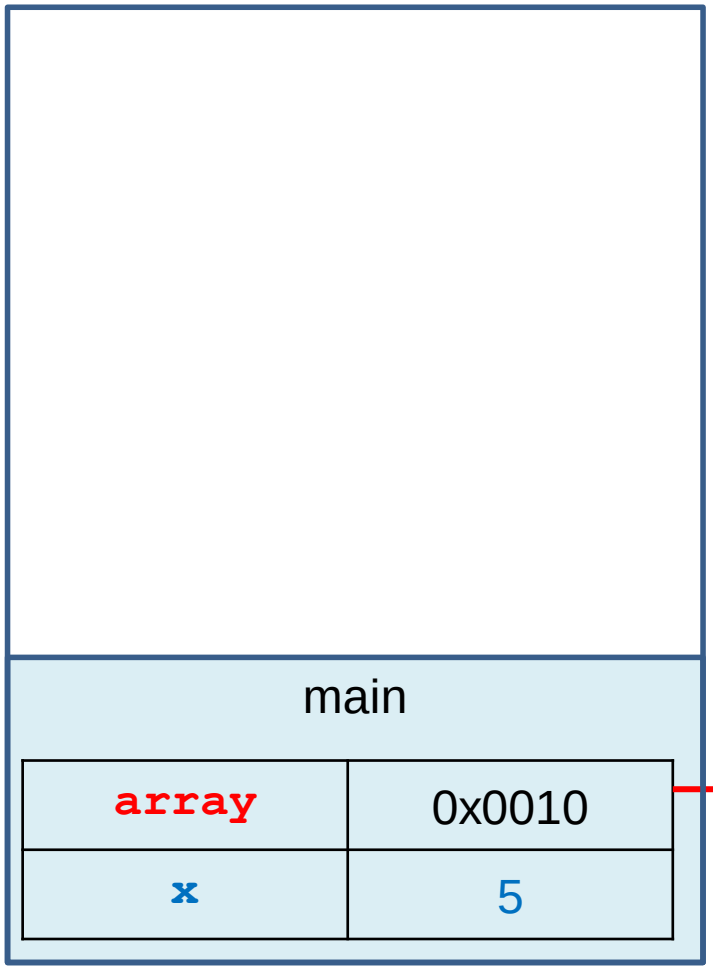


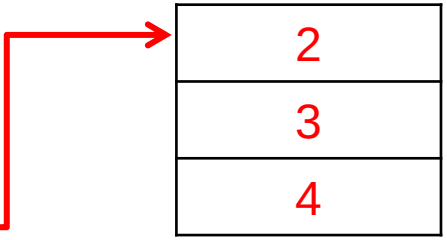
Πέρασμα παραμέτρων

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



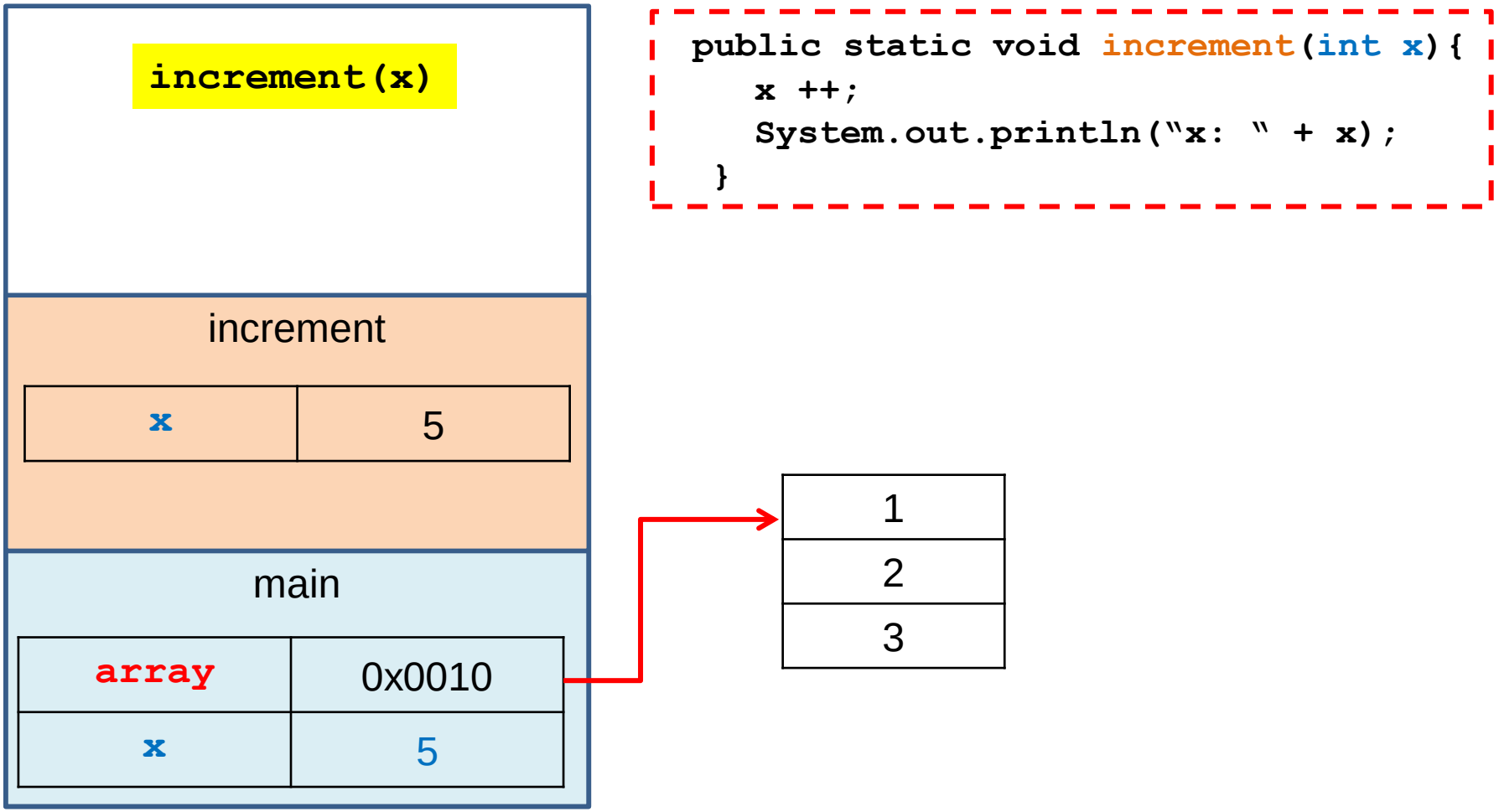
Πέρασμα παραμέτρων





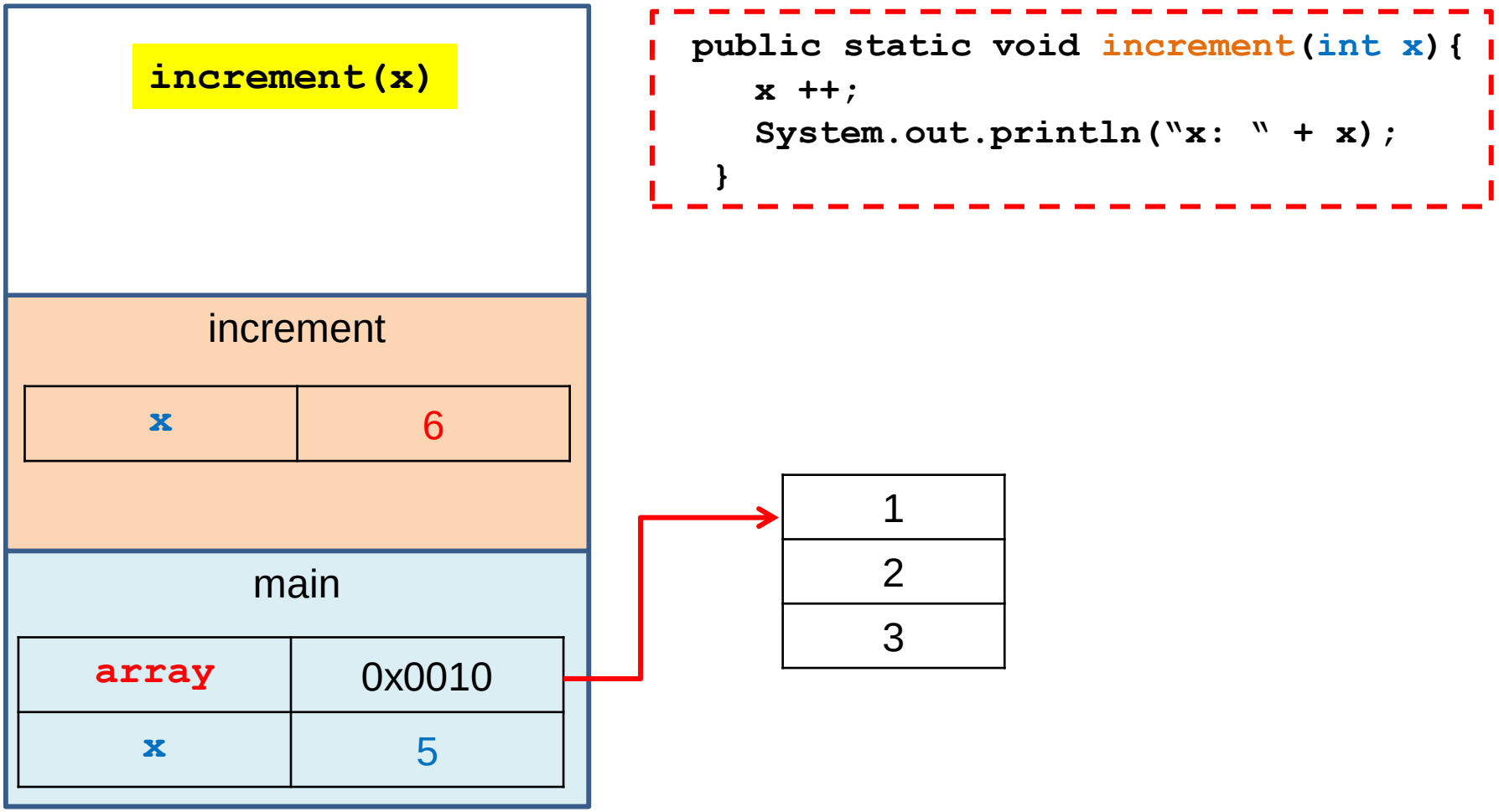
Επιστρέφοντας από την μέθοδο increment οι αλλαγές στον πίνακα παραμένουν.

Πέρασμα παραμέτρων



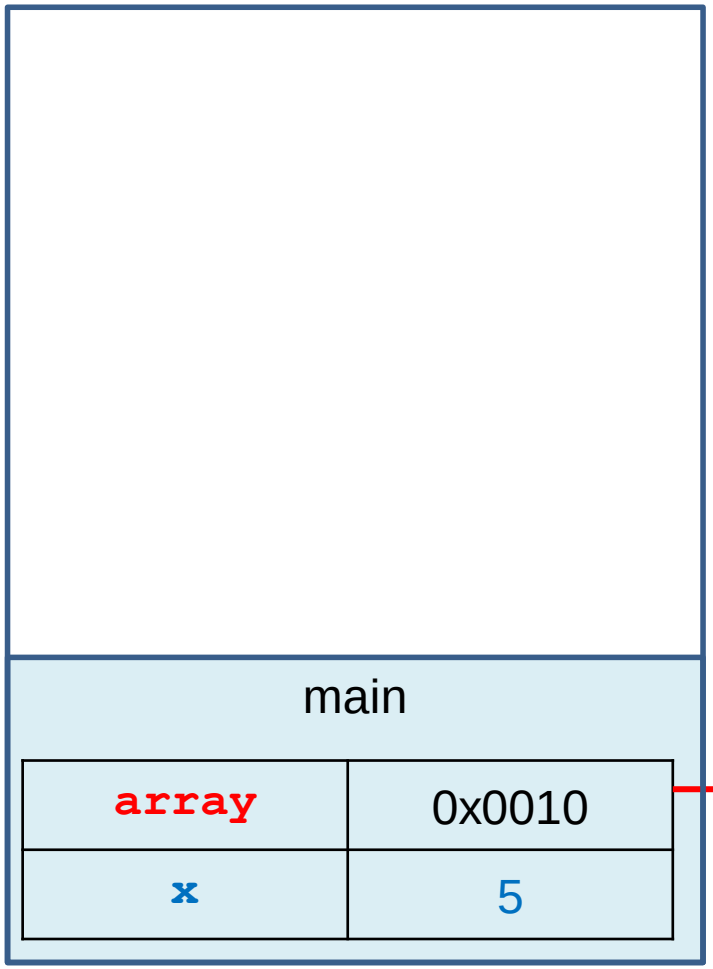

```
public static void increment(int x){
    x ++;
    System.out.println("x: " + x);
 }
```

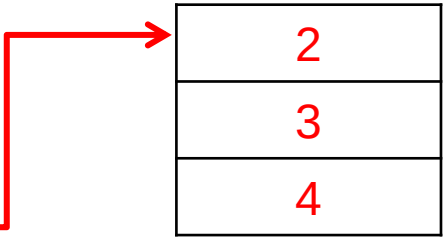
Πέρασμα παραμέτρων




```
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. .

```
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.name = this.name;
        other.number = this.number;
    }

}
```

Μια άλλη υλοποίηση της copier

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

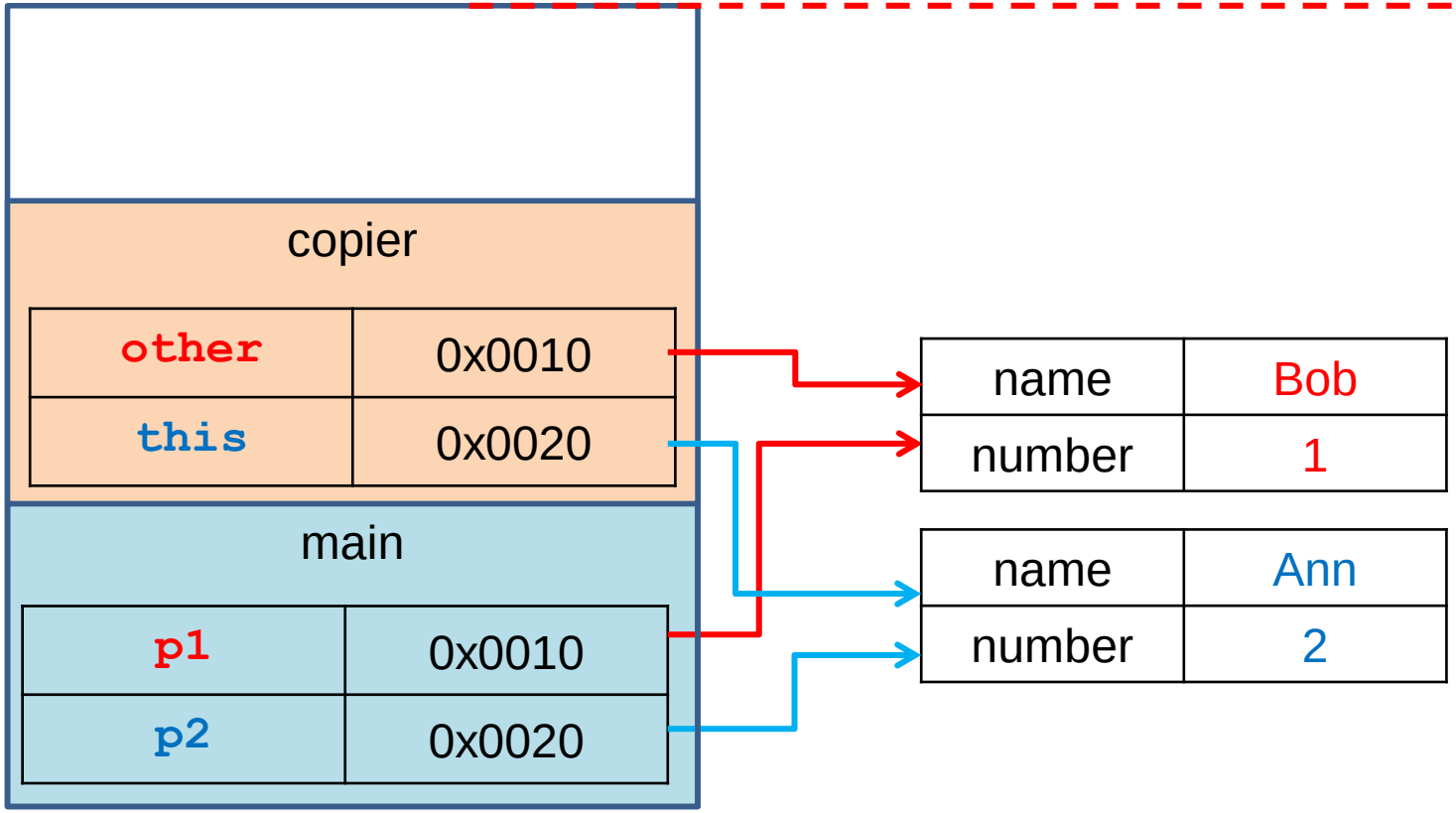
```
public class ClassParameterDemo
{
    public static void main(String[] args)
    {
        Person p1 = new Person("Bob", 1);
        Person p2 = new Person("Ann", 2);
        p2.copier(p1);
        System.out.println(p1);
     }
}
```

Τι θα τυπώσει?

Εξέλιξη του προγράμματος

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

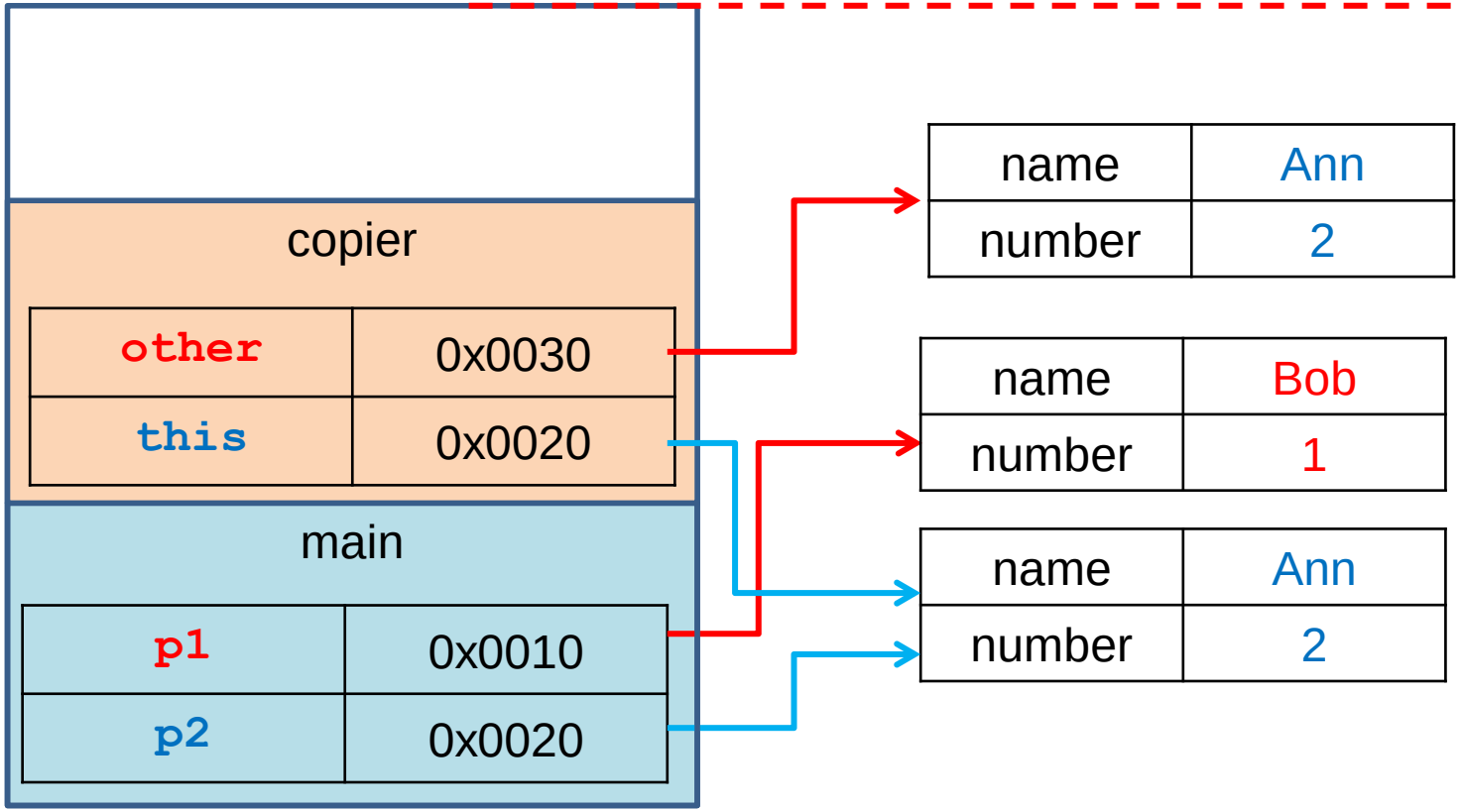
```
public void copier( Person other) {
 other = new Person(this.name, this.number);
}
```



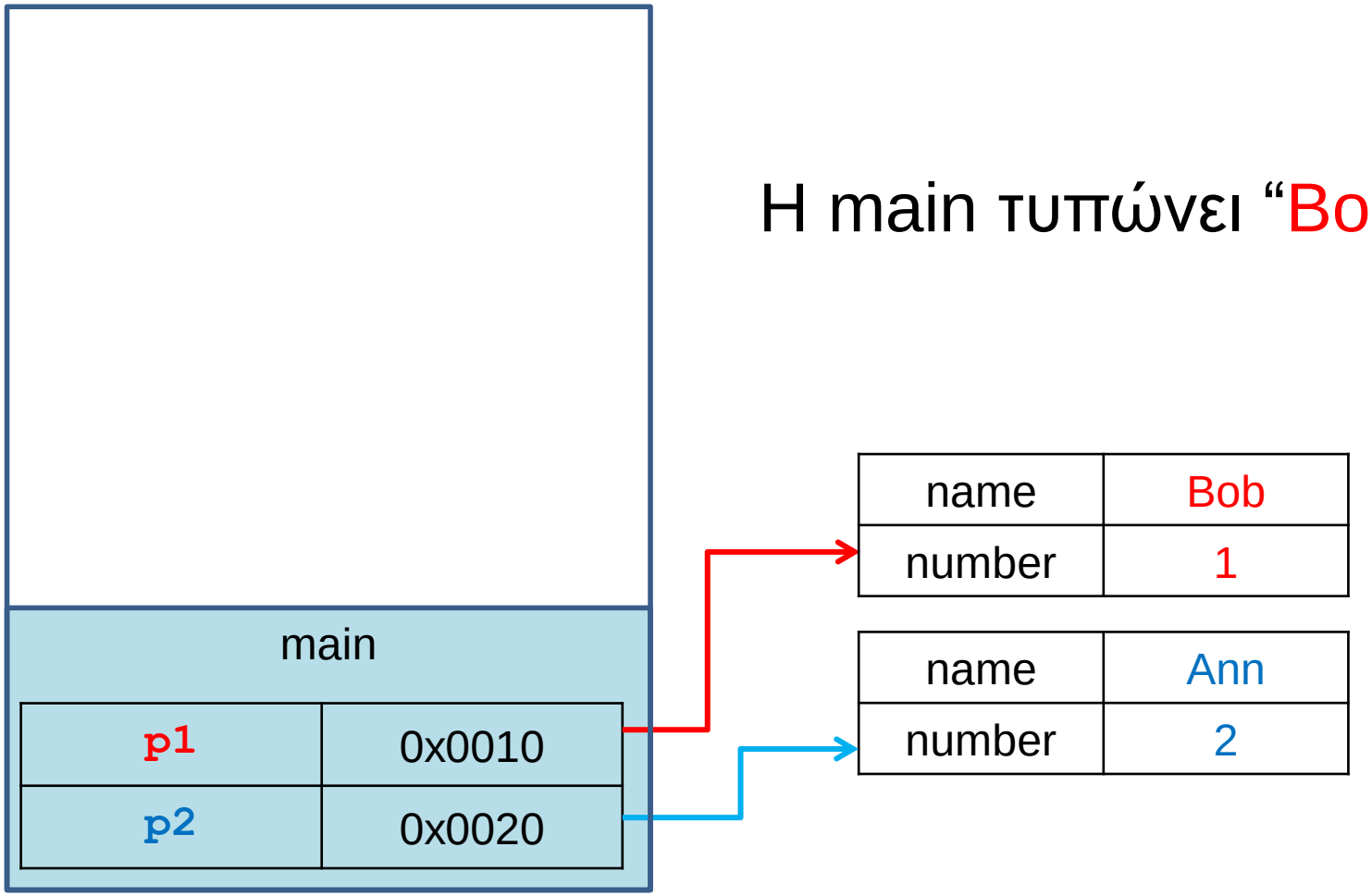
Εξέλιξη του προγράμματος

```
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.

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

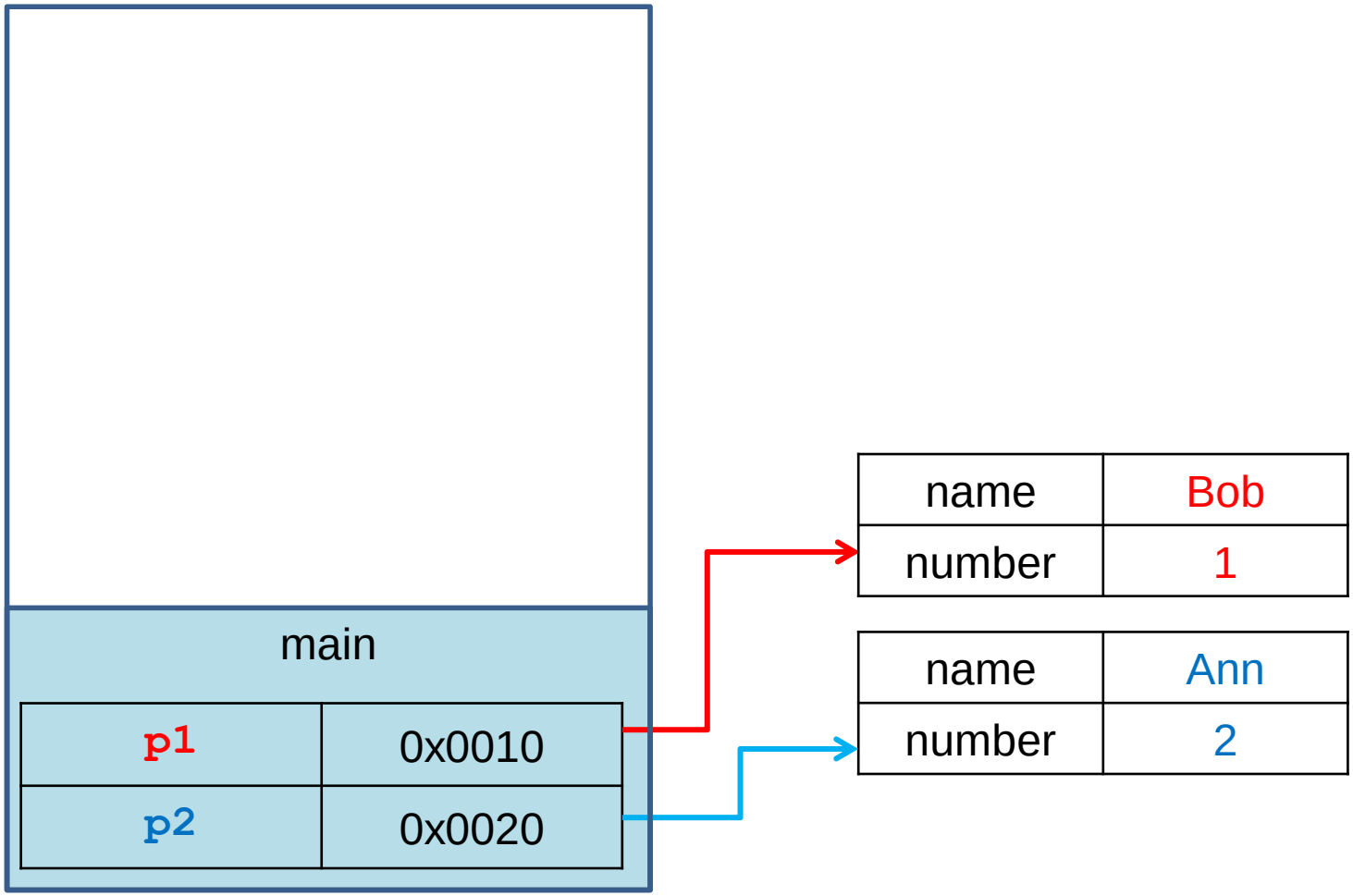
```
public class Person
{
    private String name;
    private int number;

    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);
     }
}
```

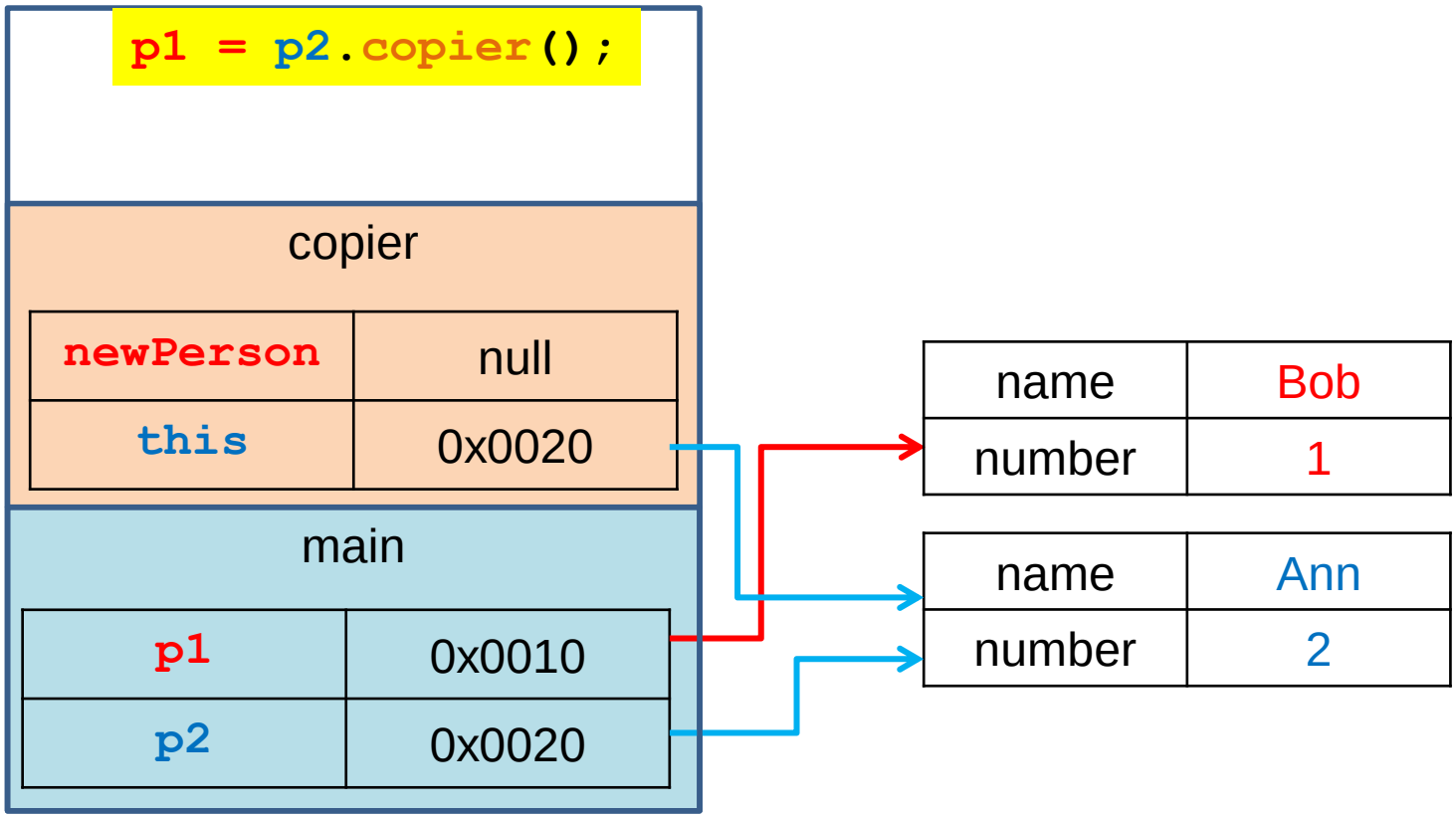
Τι θα τυπώσει?

Εξέλιξη του προγράμματος



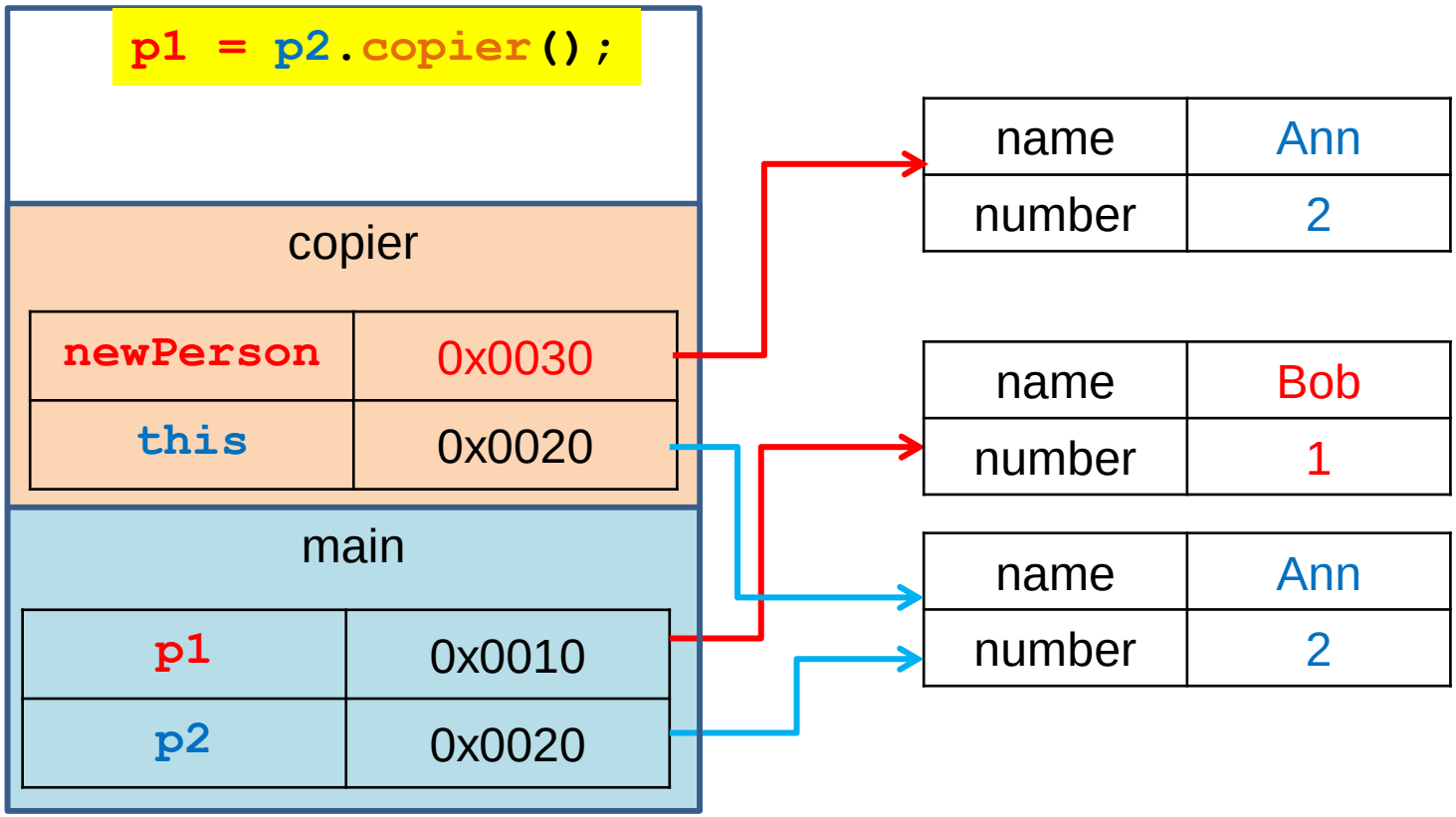
Εξέλιξη του προγράμματος

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



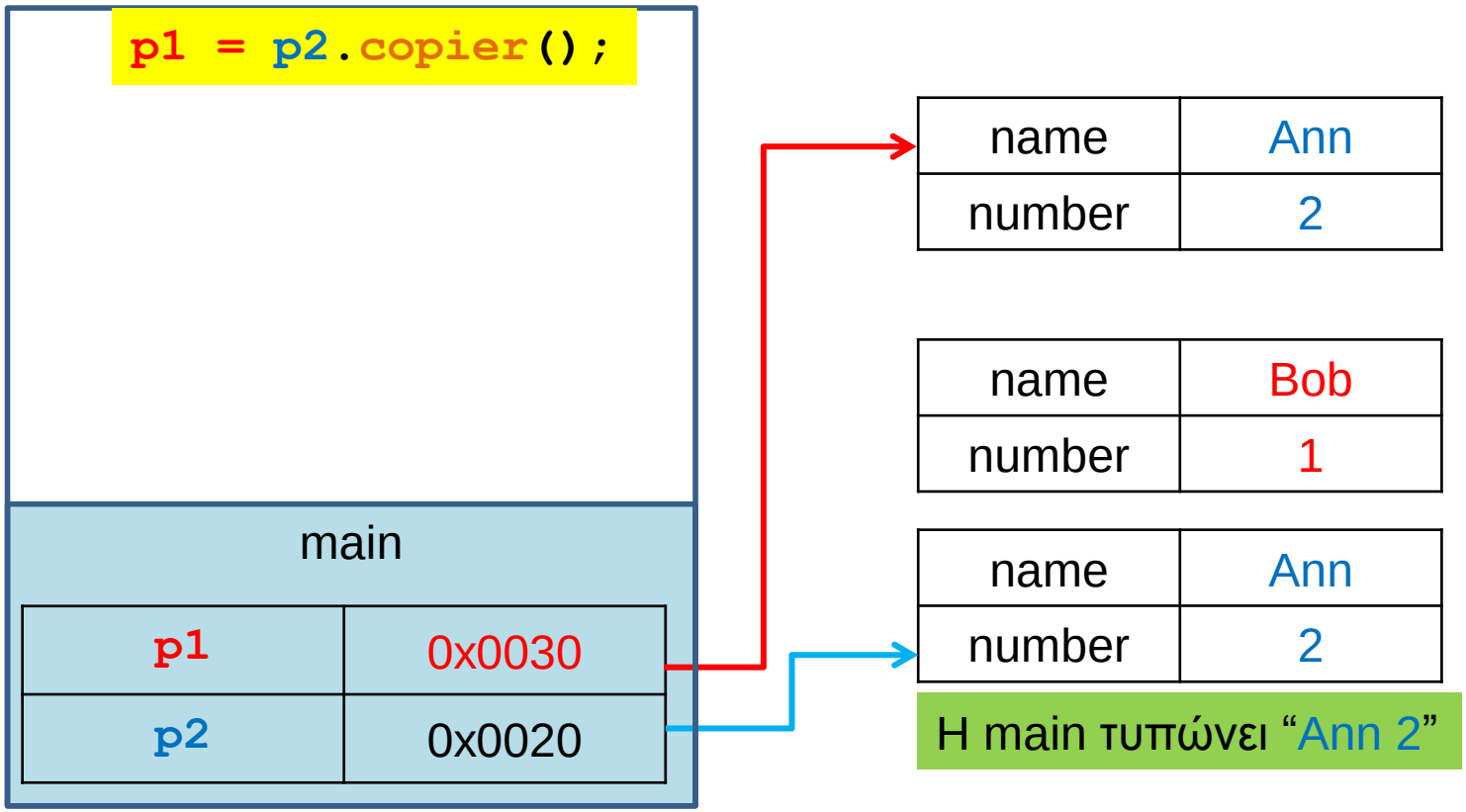
Εξέλιξη του προγράμματος

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



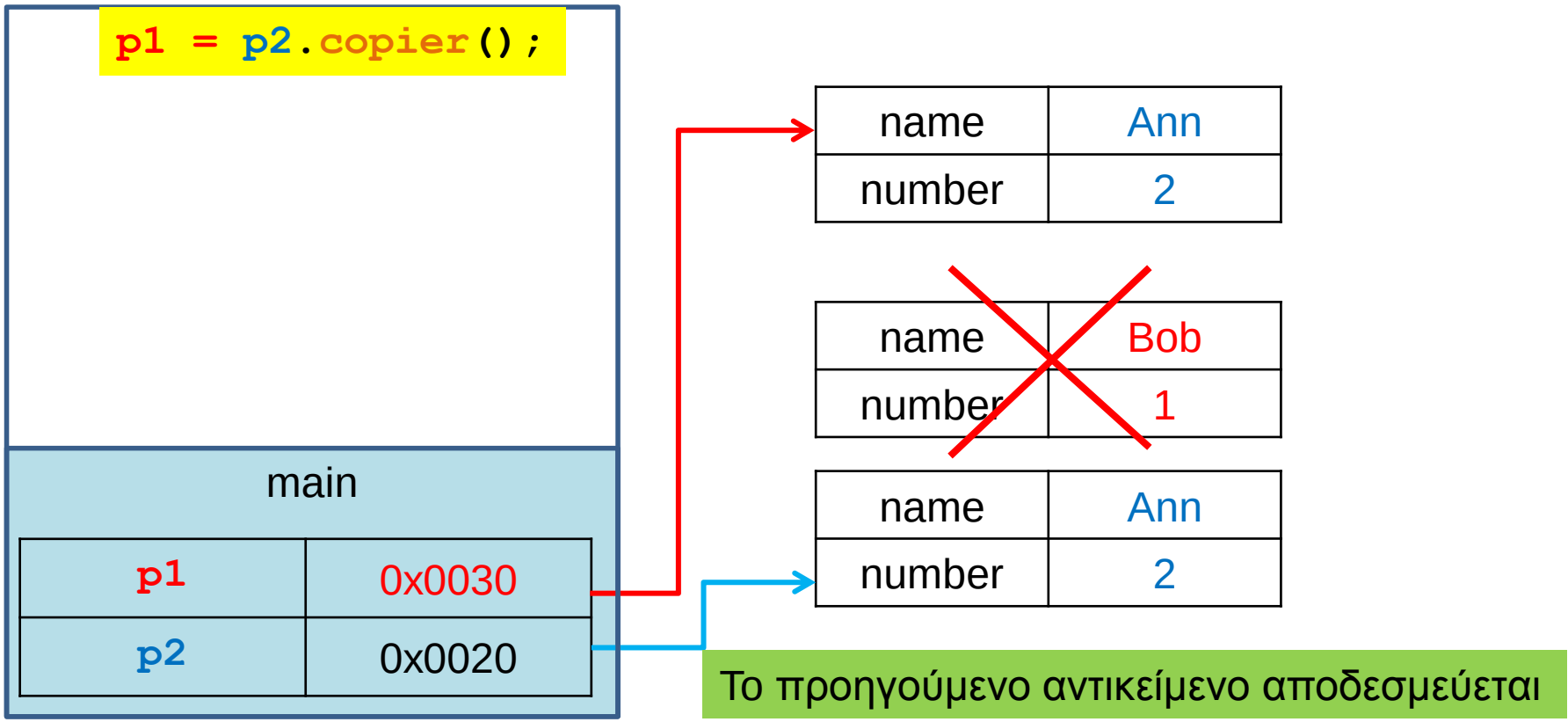
Εξέλιξη του προγράμματος

```
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

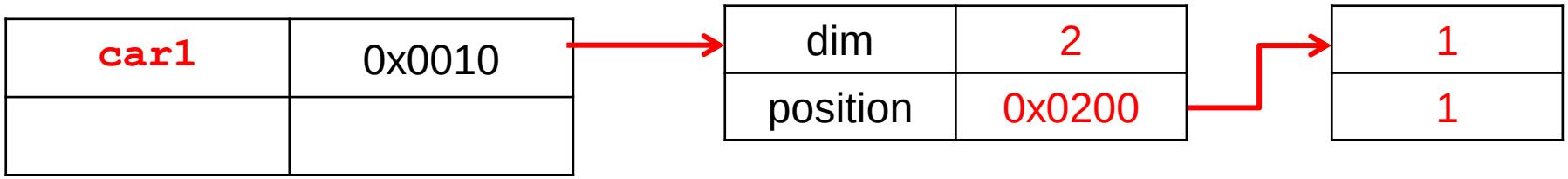
- Η μέθοδος copier όπως την ορίσαμε πριν δημιουργεί ένα καινούριο αντικείμενο που είναι αντίγραφο αυτού που έκανε την κλήση.
- Στην περίπτωση μας το αντικείμενο έχει μόνο πεδία που είναι πρωταρχικού τύπου ή μη μεταλλάξιμα αντικείμενα. Γενικά ένα αντικείμενο μπορεί να έχει ως πεδία άλλα αντικείμενα (δηλαδή 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 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);
    }
}
```

Τι θα τυπώσει η main?

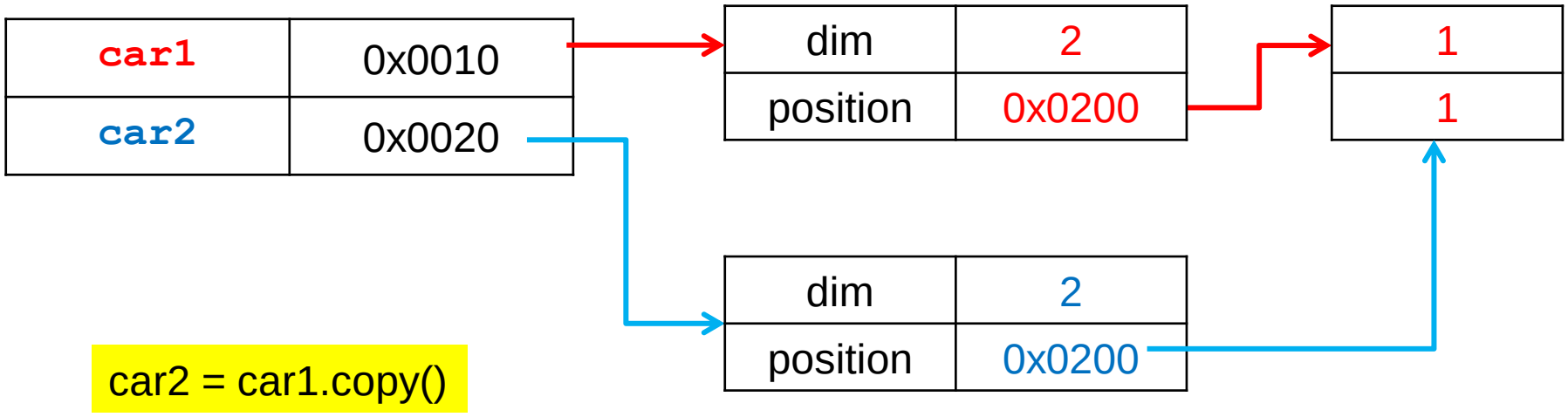
Ρηχά Αντίγραφα

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



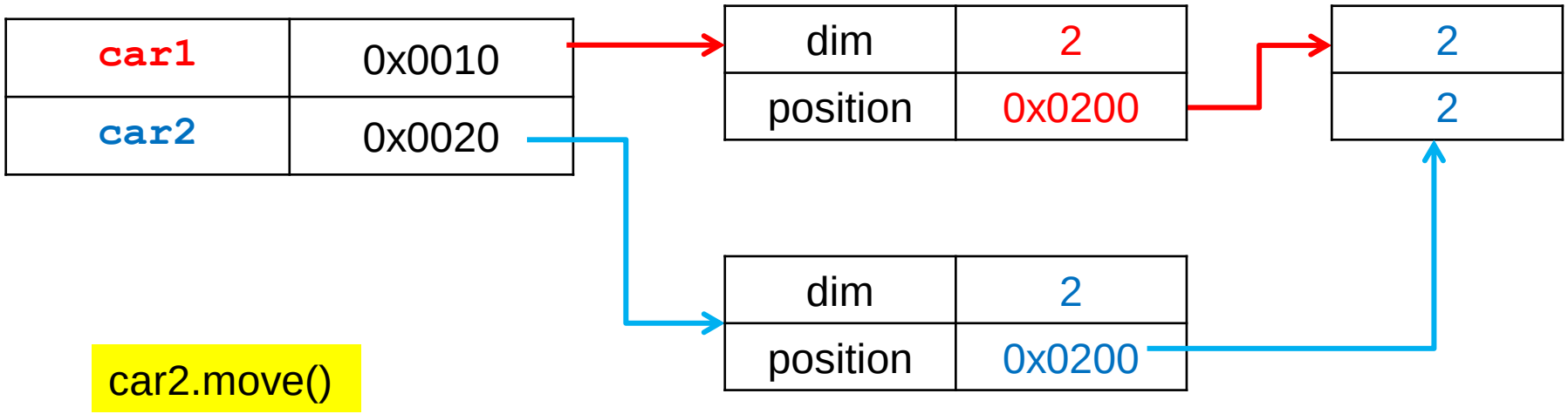
Ρηχά Αντίγραφα

- 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

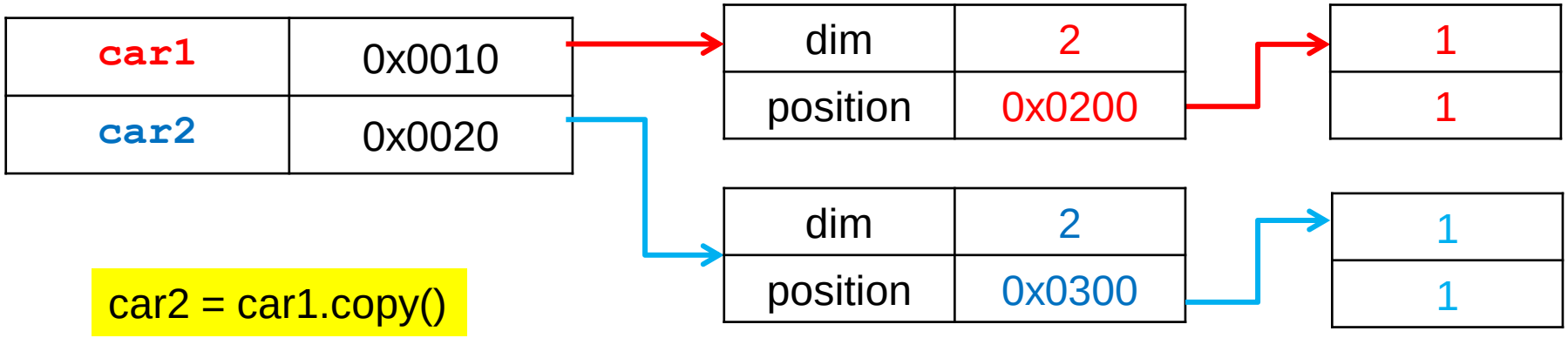


Μετακινείται και το car1 αλλά αυτό δεν είναι επιθυμητό.

Βαθύ αντίγραφο

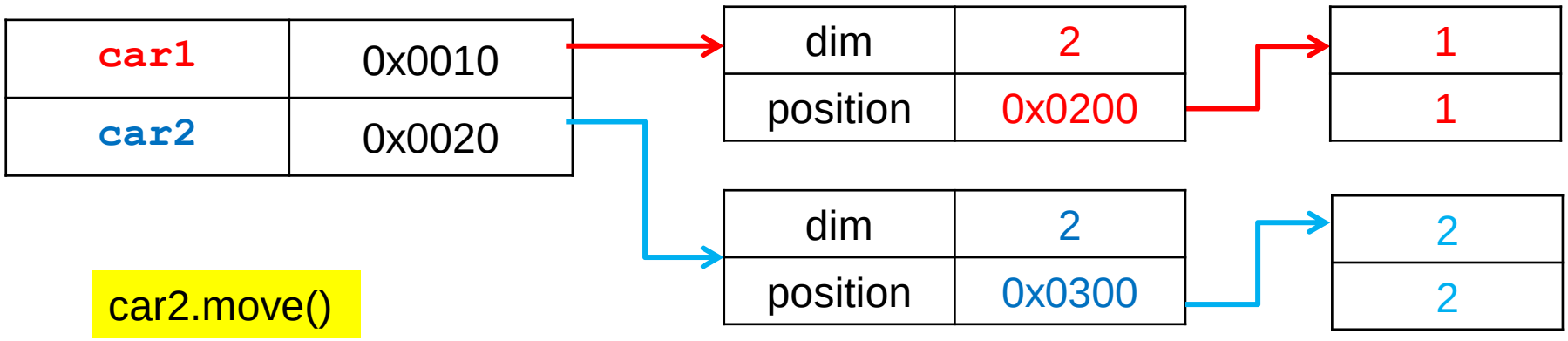
- Τις περισσότερες φορές θέλουμε να κάνουμε ένα βαθύ αντίγραφο του αντικειμένου, όπου για κάθε αντικείμενο μέσα στο αντίγραφο δεσμεύουμε νέα μνήμη

```
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

Παραδείγματα

- Τι γίνεται αν έχουμε ένα constructor που παίρνει όρισμα ένα πίνακα?
  - `public Car(int[] position)`

- Τι γίνεται αν στο ρηχό αντίγραφο κάνουμε τον πίνακα null?

Copy Constructor

- Ένας Constructor που παίρνει σαν όρισμα ένα αντικείμενο του ίδιου τύπου και δημιουργεί ένα αντίγραφο
  - `public Car(Car other)`

- Ο copy constructor έχει δύο λειτουργίες:
  - Δεσμεύει τη μνήμη για το αντικείμενο
  - Αντιγράφει τις τιμές του αντικειμένου-ορίσματος.
- Πάντα πρέπει να δημιουργούμε ένα βαθύ αντίγραφο του αντικειμένου

Copy Constructor για την Car

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

Δημιουργεί βαθύ αντίγραφο:
Δεσμεύουμε καινούριο πίνακα και αντιγράφουμε μία-μία τις τιμές

Κλήση:

```
Car car1 = new Car(2);
Car car2 = new Car(car1);
```

Φωλιασμένος Copy Constructor

- Αν μια κλάση έχει πεδία αντικείμενα από μία άλλη κλάση, τότε όταν καλούμε τον copy constructor θα πρέπει να έχουμε ορίσει copy constructor και για τις κλάσεις των αντικειμένων-πεδίων.

Παράδειγμα

```
public class CarDriver
{
    private int position;
    private Person driver;

    public CarDriver(CarDriver other){
        this.position = other.position;
        driver = new Person(other.driver);
    }
}
```

Καλεί την copy constructor της Person

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

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

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

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

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

    public boolean equals(Person other){
        return (this.name.equals(other.name) && this.number == other.number;
    }
}
```

Φωλιασμένη equals

```
public class CarDriver
{
      private int position;
      private Person driver;

      public CarDriver(CarDriver other){
         this.position = other.position;
         driver = new Person(other.driver);
      }

       public boolean equals(CarDriver other){
         return this.driver.equals(other.driver)
                && this.position == other.position;
       }
}
```

Καλεί την equals της Person

Φωλιασμένη toString()

```
public class CarDriver
{
      private int position;
      private Person driver;

      public CarDriver(CarDriver other){
          this.position = other.position;
          driver = new Person(other.driver);
      }

        public boolean equals(CarDriver other){
            return this.driver.equals(other.driver)
                && this.position == other.position;
        }

        public String toString(){
            return driver + " " + position;
        }
}
```

Καλεί την toString της Person

Πίνακες από αντικείμενα

- Όπως ορίζουμε πίνακες από πρωταρχικούς τύπους μπορούμε να ορίσουμε και πίνακες από αντικείμενα
  - `Person[] array = new Person[3];`
  - Ορίζει ένα πίνακα με τρία αντικείμενα τύπου Person
  - Ουσιαστικά ένα πίνακα με αναφορές.

- Όταν ορίζουμε ένα πίνακα από αντικείμενα πρέπει να είμαστε προσεκτικοί να δεσμεύουμε σωστά τη μνήμη.

Παράδειγμα

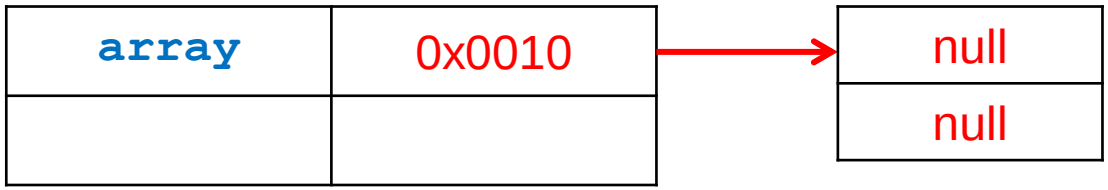
```
Person[] array;
```

array	null

- Η εντολή αυτή θα δημιουργήσει μια μεταβλητή με το όνομα array η οποία κάποια στιγμή θα δείχνει σε ένα πίνακα με Person. Για την ώρα είναι null.

Παράδειγμα

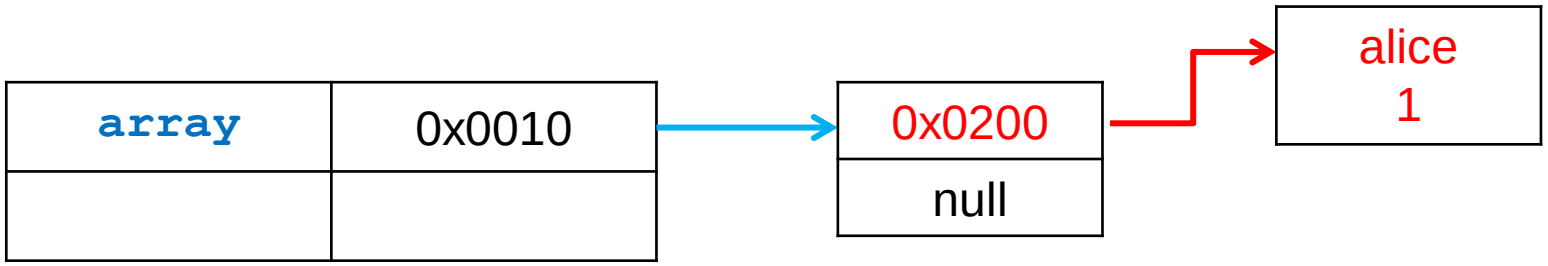
```
Person[] array;
array = new Person[2];
```



- Η εντολή new θα δεσμεύσει δύο θέσεις μνήμης στο heap για να κρατήσουν δύο αναφορές τύπου Person. Εφόσον δεν έχουμε δημιουργήσει τις μεταβλητές ακόμη, αυτές θα είναι null.

Παράδειγμα

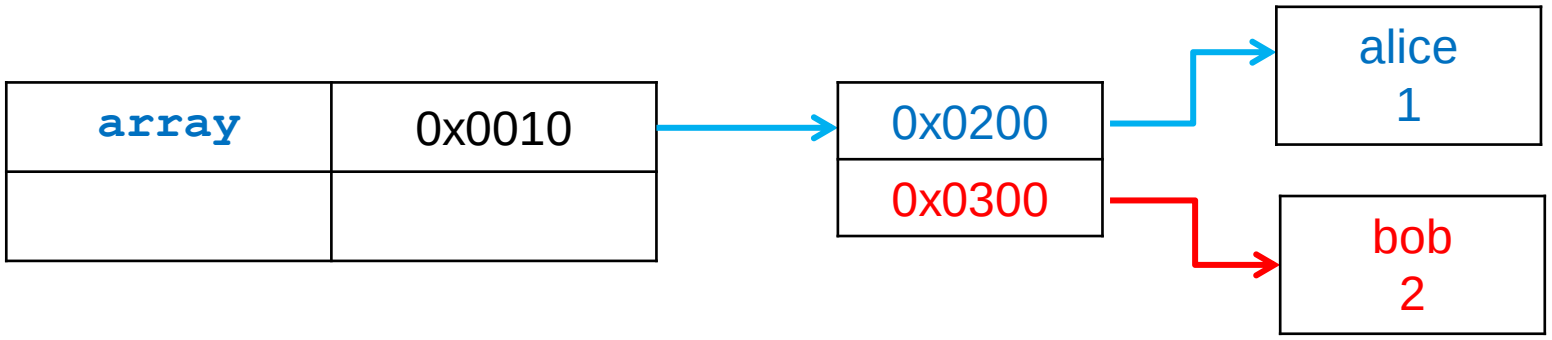
```
Person[] array;
array = new Person[2];
array[0] = new Person("alice", 1);
```



- Η νέα εντολή new θα δεσμεύσει χώρο για ένα Person. Δημιουργείται το αντικείμενο και η αναφορά αποθηκεύεται στην πρώτη θέση του πίνακα array.

Παράδειγμα

```
Person[] array;
array = new Person[2];
array[0] = new Person("alice", 1);
array[1] = new Person("bob", 1);
```



- Η νέα εντολή new θα δεσμεύσει χώρο για άλλο ένα Person. Δημιουργείται το αντικείμενο και η αναφορά αποθηκεύεται στην δεύτερη θέση του πίνακα array.

Πίνακες από πίνακες

- Οι δισδιάστατοι πίνακες είναι ουσιαστικά πίνακες από αντικείμενα, όπου τα αντικείμενα είναι πάλι πίνακες
- Π.χ., έτσι δεσμεύουμε πίνακα ακεραίων 10×10

```
int[][] array;
array = new int[10][];
for (int i=0; i<10; i++){
     array[i] = new int[10];
}
```

Πίνακες από πίνακες

- Μπορεί ο δισδιάστατος μας πίνακας να είναι ασύμμετρος.
- Π.χ., έτσι ορίζουμε ένα διαγώνιο πίνακα.

```
int[][] array;
array = new int[10][];
for (int i=0; i<10; i++){
     array[i] = new int[i+1];
}
```