

ΤΕΧΝΙΚΕΣ ΑΝΤΙΚΕΙΜΕΝΟΣΤΡΑΦΟΥΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ

Αναφορές

Μέθοδοι που επιστρέφουν αντικείμενα

Copy Constructor

Deep and Shallow Copies

```
class Person
{
    private String name;

    public Person(String name) {
        this.name = name;
    }

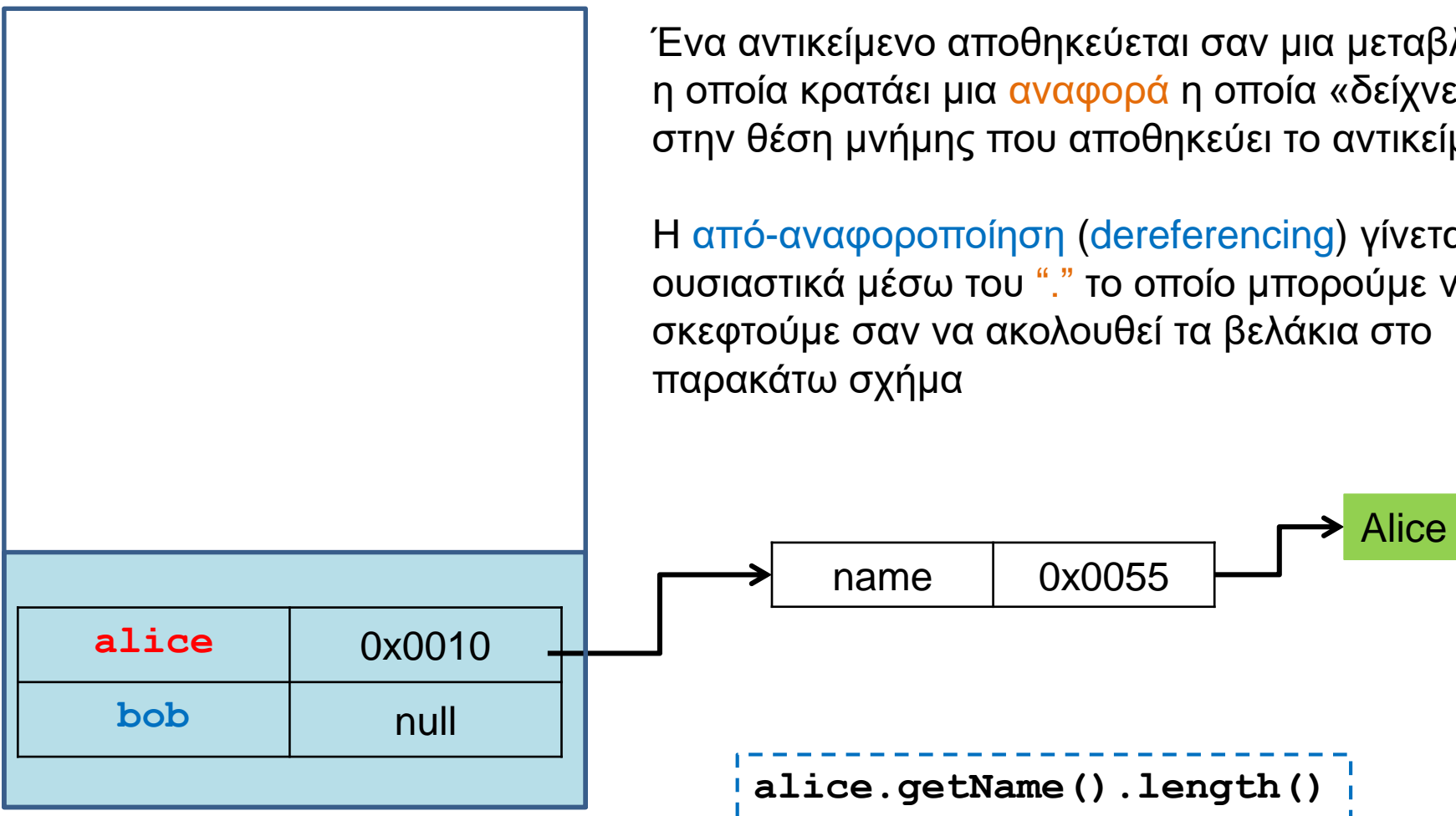
    public String getName() {
        return name;
    }
}
```

```
class PersonTest
{
    public static void main(String args[])
    {
        Person alice = new Person("Alice");
        Person bob;
        System.out.println(alice.getName());
        System.out.println(alice.getName().length());
    }
}
```

Dereferencing

Ένα αντικείμενο αποθηκεύεται σαν μια μεταβλητή η οποία κρατάει μια αναφορά η οποία «δείχνει» στην θέση μνήμης που αποθηκεύει το αντικείμενο.

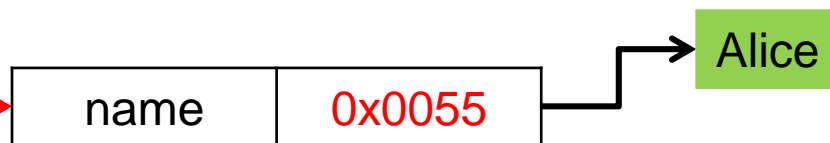
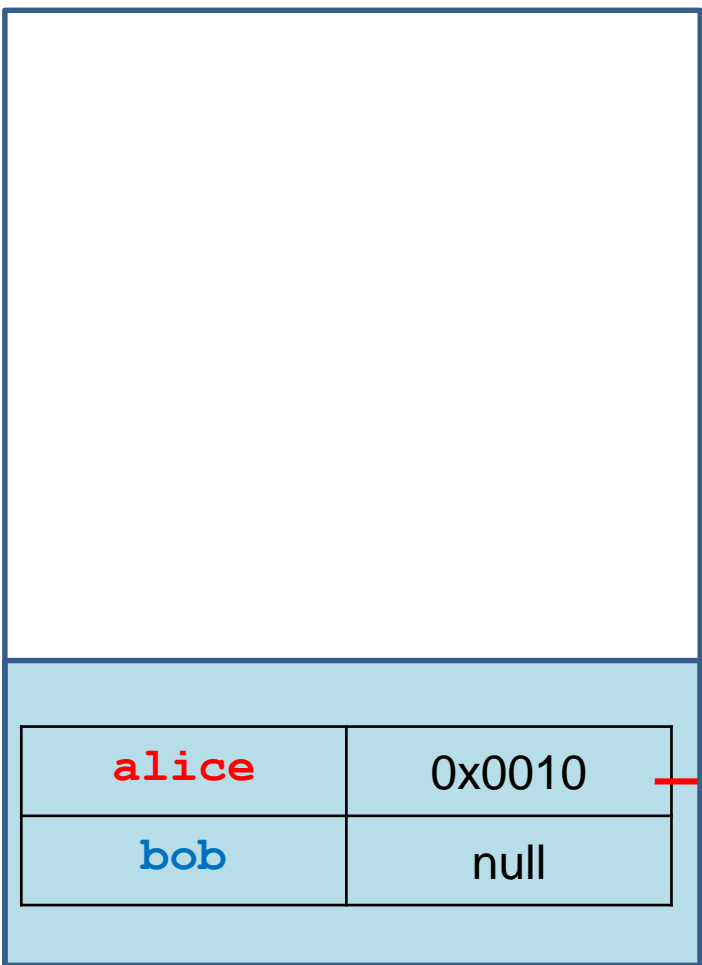
Η από-αναφοροποίηση (dereferencing) γίνεται ουσιαστικά μέσω του "." το οποίο μπορούμε να σκεφτούμε σαν να ακολουθεί τα βελιάκια στο παρακάτω σχήμα



Dereferencing

Ένα αντικείμενο αποθηκεύεται σαν μια μεταβλητή η οποία κρατάει μια αναφορά η οποία «δείχνει» στην θέση μνήμης που αποθηκεύει το αντικείμενο.

Η από-αναφοροποίηση (dereferencing) γίνεται ουσιαστικά μέσω του "." το οποίο μπορούμε να σκεφτούμε σαν να ακολουθεί τα βελάκια στο παρακάτω σχήμα

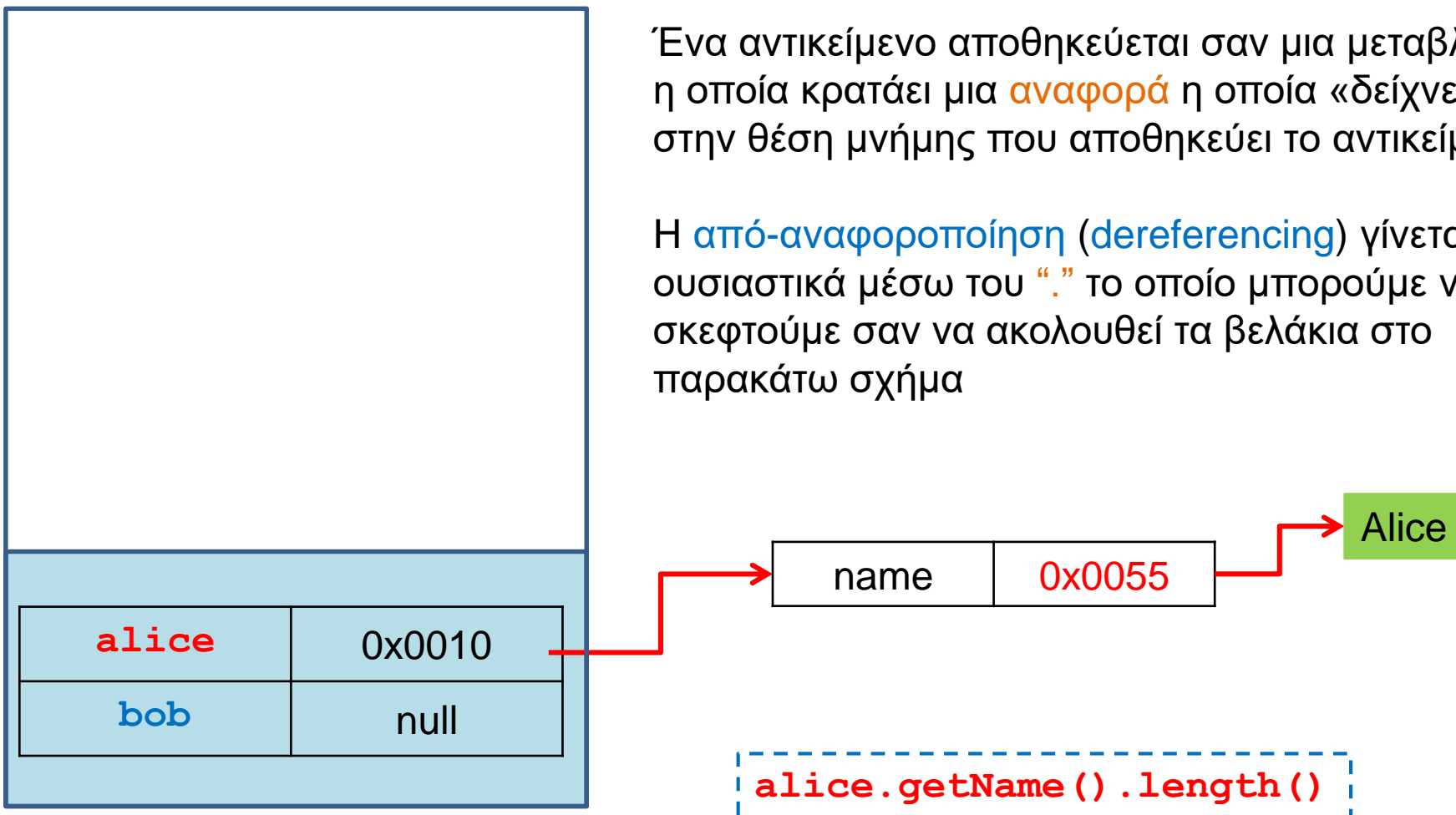


```
alice.getName().length()
```

Dereferencing

Ένα αντικείμενο αποθηκεύεται σαν μια μεταβλητή η οποία κρατάει μια αναφορά η οποία «δείχνει» στην θέση μνήμης που αποθηκεύει το αντικείμενο.

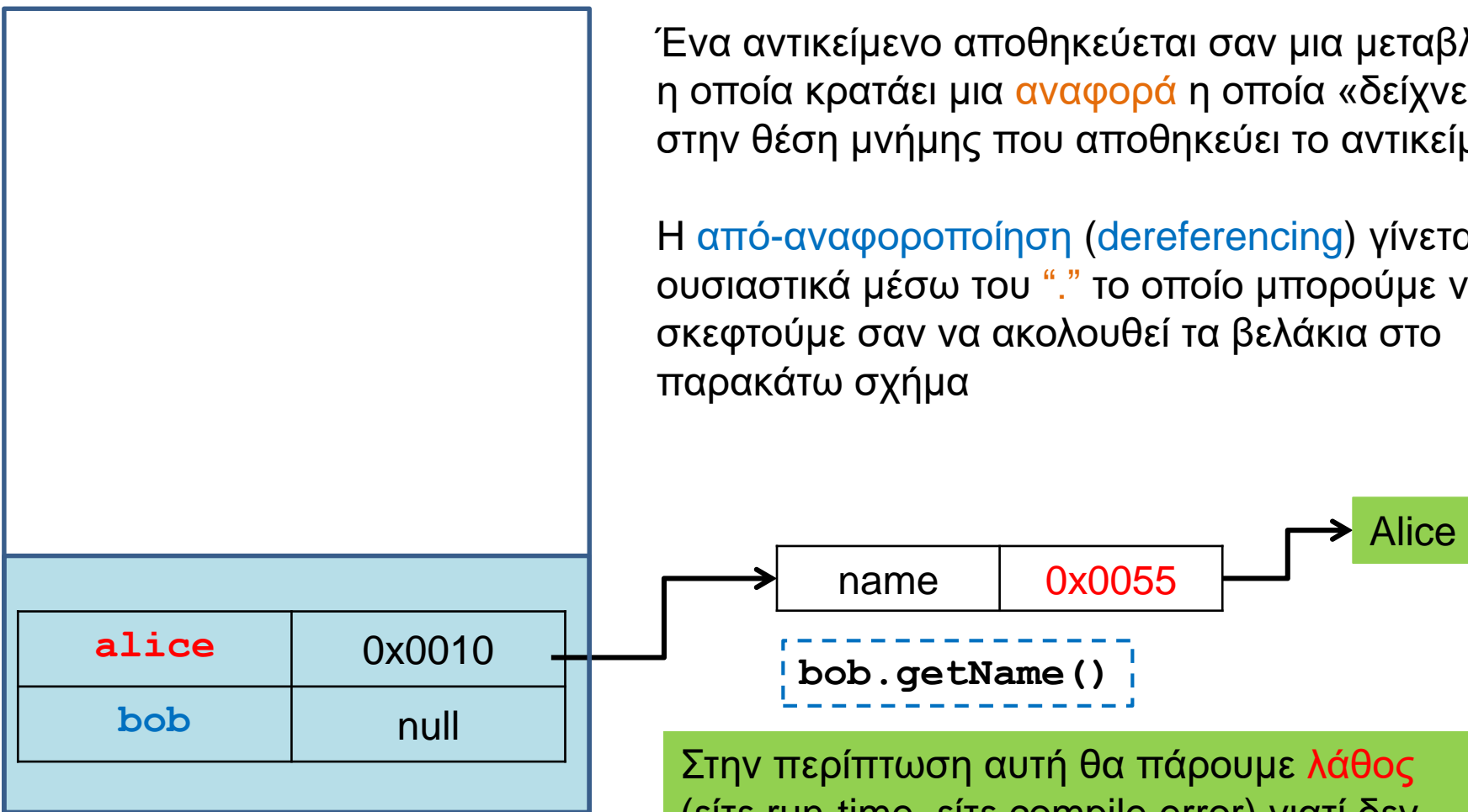
Η από-αναφοροποίηση (dereferencing) γίνεται ουσιαστικά μέσω του "." το οποίο μπορούμε να σκεφτούμε σαν να ακολουθεί τα βελάκια στο παρακάτω σχήμα



Dereferencing

Ένα αντικείμενο αποθηκεύεται σαν μια μεταβλητή η οποία κρατάει μια αναφορά η οποία «δείχνει» στην θέση μνήμης που αποθηκεύει το αντικείμενο.

Η από-αναφοροποίηση (dereferencing) γίνεται ουσιαστικά μέσω του "." το οποίο μπορούμε να σκεφτούμε σαν να ακολουθεί τα βελάκια στο παρακάτω σχήμα



Στην περίπτωση αυτή θα πάρουμε **λάθος** (είτε run-time, είτε compile error) γιατί δεν υπάρχει διεύθυνση να ακολουθήσουμε

```
class Person
```

```
{  
    private String name;  
  
    public Person(String name) {  
        this.name = name;  
    }  
  
    public String getName () {  
        return name;  
    }  
}
```

```
class Car
```

```
{  
    private int position = 0;  
    private Person driver;  
  
    public Car(int position, Person driver) {  
        this.position = position;  
        this.driver = driver;  
    }  
  
    public Person getDriver () {  
        return driver;  
    }  
}
```

```
class MovingCarDriver1
```

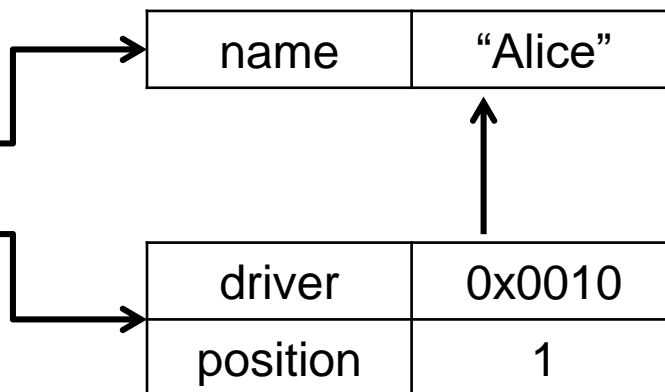
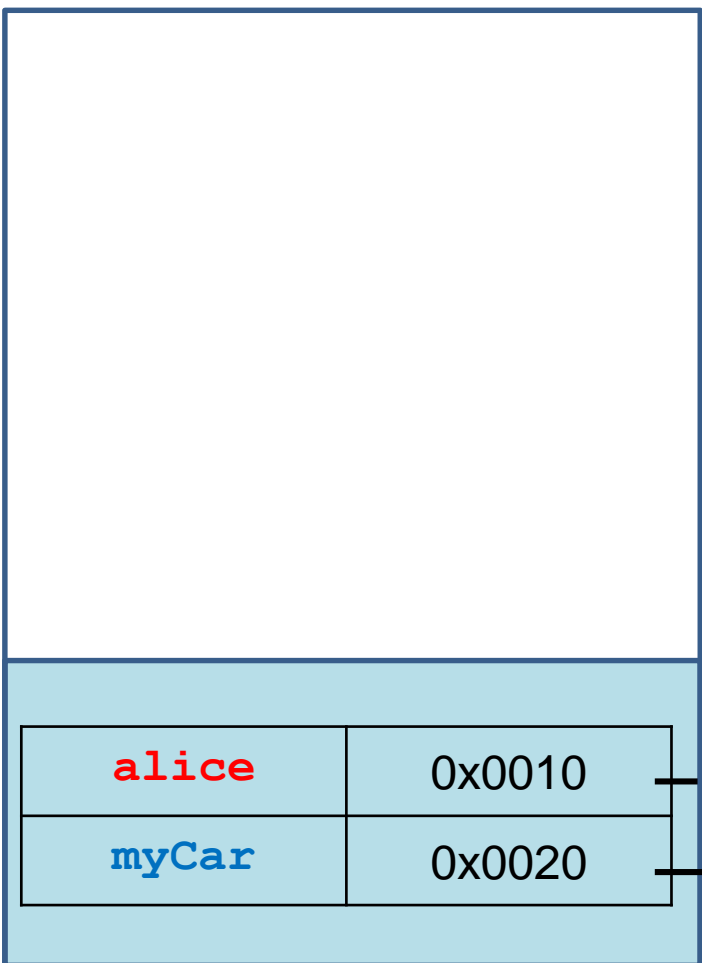
```
{  
    public static void main(String args[])  
    {  
        Person alice = new Person("Alice");  
        Car myCar = new Car(1, alice);  
        System.out.println(myCar.getDriver().getName());  
    }  
}
```

Dereferencing

Στην περίπτωση αυτή έχουμε ένα αντικείμενο μέσα σε ένα άλλο αντικείμενο.
Η μέθοδος `getDriver()` επιστρέφει αντικείμενο `Person`

Έχουμε αλυσιδωτή πρόσβαση σε αναφορές

```
myCar.getDriver().getName()
```

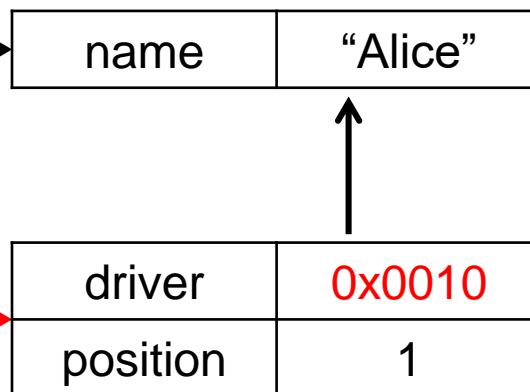
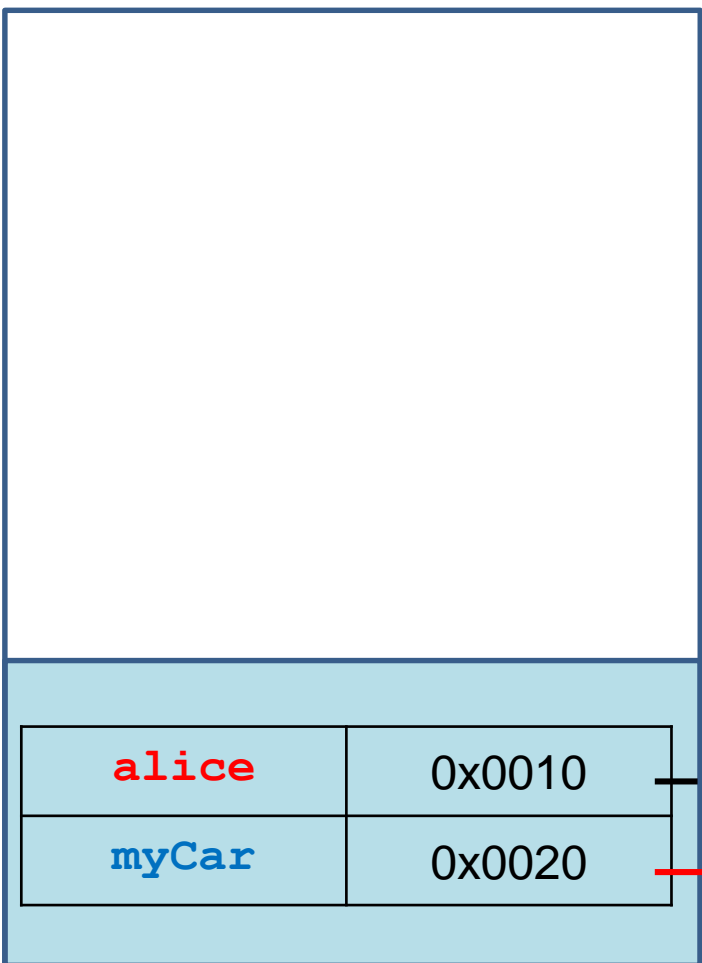


Dereferencing

Στην περίπτωση αυτή έχουμε ένα αντικείμενο μέσα σε ένα άλλο αντικείμενο.
Η μέθοδος `getDriver()` επιστρέφει αντικείμενο `Person`

Έχουμε αλυσιδωτή πρόσβαση σε αναφορές

```
myCar.getDriver().getName()
```

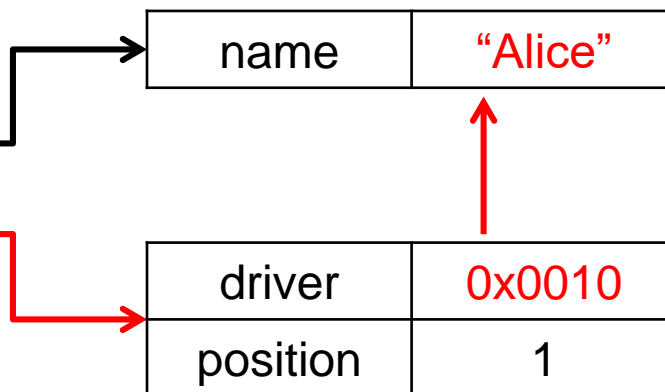
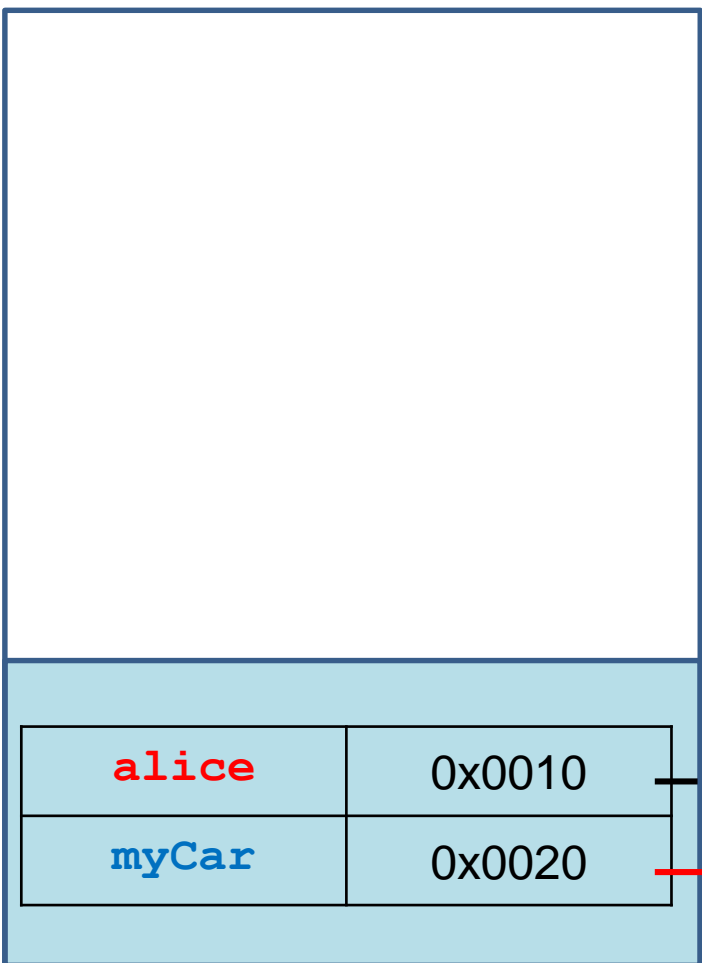


Dereferencing

Στην περίπτωση αυτή έχουμε ένα αντικείμενο μέσα σε ένα άλλο αντικείμενο.
Η μέθοδος `getDriver()` επιστρέφει αντικείμενο `Person`

Έχουμε αλυσιδωτή πρόσβαση σε αναφορές

```
myCar.getDriver().getName()
```



```
class Person
```

```
{  
    private String name;  
  
    public Person(String name){  
        this.name = name;  
    }  
  
    public String getName(){  
        return name;  
    }  
}
```

```
class Car
```

```
{  
    private int position = 0;  
    private Person driver;  
  
    public Car(int position, String name){  
        this.position = position;  
        this.driver = new Person(name);  
    }  
  
    public String getDriverName(){  
        return driver.getName();  
    }  
}
```

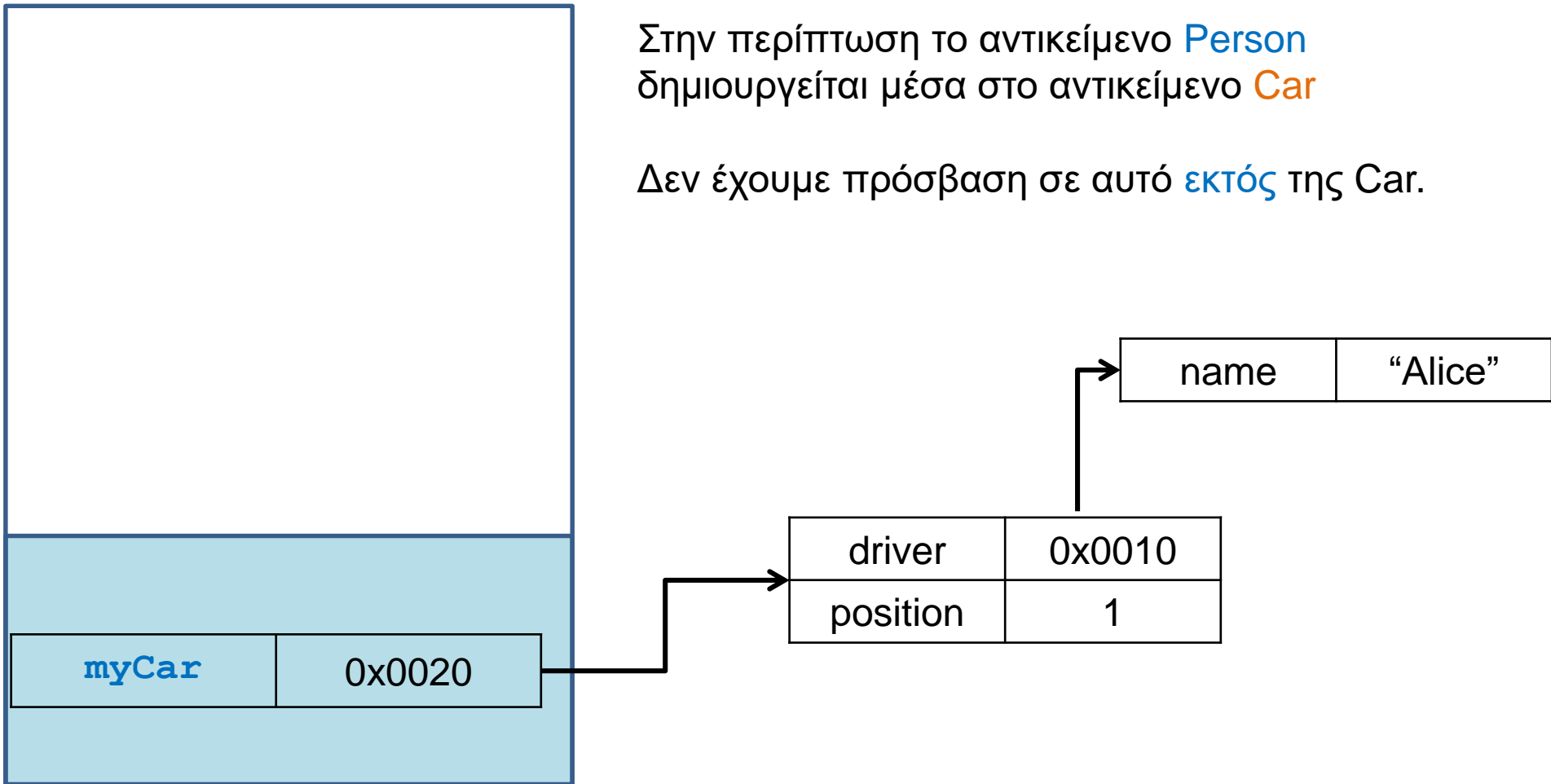
```
class MovingCarDriver2
```

```
{  
    public static void main(String args[])  
    {  
        Car myCar = new Car(1, "Alice");  
        System.out.println(myCar.getDriverName());  
    }  
}
```

Αντικείμενα μέσα σε αντικείμενα

Στην περίπτωση το αντικείμενο **Person** δημιουργείται μέσα στο αντικείμενο **Car**

Δεν έχουμε πρόσβαση σε αυτό **εκτός** της Car.



Σχέσεις μεταξύ κλάσεων

- Στο παράδειγμα μας έχουμε δύο διαφορετικές κλάσεις (**Person**, **Driver**) οι οποίες συσχετίζονται μεταξύ τους με διαφορετικούς τρόπους.
- Μπορεί να υπάρχουν πολλές διαφορετικές σχέσεις μεταξύ κλάσεων.
 - Στην περίπτωση μας, η μία κλάση ορίζεται χρησιμοποιώντας αντικείμενα της άλλης
- Αυτού του είδους τη σχέση την λέμε σχέση **σύνθεσης**
 - Μερικές φορές την ξεχωρίζουμε σε σχέση **σύνθεσης** (composition) και **συνάθροισης** (aggregation).

Σχέσεις κλάσεων

- Όταν έχουμε **κλάσεις** που **έχουν αντικείμενα άλλων κλάσεων** ένα θέμα που προκύπτει είναι πότε και πού θα γίνεται η **δημιουργία των αντικειμένων** και πότε η καταστροφή τους
 - Πιο σημαντικό σε γλώσσες που δεν έχουν garbage collector.
- Π.χ., τα αντικείμενα τύπου **Person** στο παράδειγμα **MovingCarDriver2** **δημιουργούνται μέσα** στην κλάση **Car**, και καταστρέφονται μέσα στην **Car**, ή αν το αντικείμενο **Car** καταστραφεί.
- Τα αντικείμενα τύπου **Person** που χρησιμοποιούνται στην **MovingCarDriver1** **δημιουργούνται εκτός της κλάσης** και μπορεί να υπάρχουν αφού καταστραφεί η κλάση.
- Συχνά οι σχέσεις του δεύτερου τύπου λέγονται σχέσεις **συνάθροισης**, ενώ σχέσεις του πρώτου τύπου, **σύνθεσης**.

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

```
class Date
```

```
{
```

```
    private int day = 1;
```

```
    private int month = 1;
```

```
    private int year = 2015;
```

```
    private String[] monthStrings =
```

```
        {"Jan", "Feb", "Mar", "Apr", "May", "Jun",  
        "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"};
```

```
    public Date(int day, int month, int year) {
```

```
        this.day = day;
```

```
        this.month = month;
```

```
        this.year = year;
```

```
    }
```

```
    public String toString() {
```

```
        return day + " " + monthNames[month-1] + " " + year;
```

```
    }
```

```
}
```

Η κλάση Date

Θέλω η κλάση να μπορεί να μου επιστρέφει μια νέα ημερομηνία αλλά ένα χρόνο μετά. Πως μπορώ να το κάνω?


```
class Date
```

```
{
```

```
    private int day = 1;
```

```
    private int month = 1;
```

```
    private int year = 2014;
```

```
    private String[] monthStrings =
```

```
        {"Jan", "Feb", "Mar", "Apr", "May", "Jun",
```

```
        "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"};
```

```
    public Date(int day, int month, int year){
```

```
        this.day = day; this.month = month; this.year = year;
```

```
    }
```

```
    public String toString(){
```

```
        return day + " " + monthNames[month-1] + " " + year;
```

```
    }
```

```
    public Date nextYear(){
```

```
        Date nextYearDate = new Date(day, month, year+1);
```

```
        return nextYearDate;
```

```
    }
```

```
}
```

Η κλάση Date

Η κλάση nextYear() επιστρέφει ένα νέο αντικείμενο Date με την ημερομηνία ένα χρόνο μετά.

```
class DateExample
{
    public static void main(String args[]) {
        Date today = new Date(25,5,2015);
        System.out.println(today);
        Date todayNextYear = today.nextYear();
        System.out.println(todayNextYear);
    }
}
```

```
class Date
```

```
{
```

```
    private int day = 1;
```

```
    private int month = 1;
```

```
    private int year = 2014;
```

```
    private String[] monthStrings =
```

```
        {"Jan", "Feb", "Mar", "Apr", "May", "Jun",
```

```
        "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"};
```

```
    public Date(int day, int month, int year){
```

```
        this.day = day; this.month = month; this.year = year;
```

```
    }
```

```
    public String toString(){
```

```
        return day + " " + monthNames[month-1] + " " + year;
```

```
    }
```

```
    public Date nextYear(){
```

```
        return new Date(day, month, year+1);
```

```
    }
```

```
}
```

Η κλάση Date

Τι γίνεται αν η ημερομηνία είναι 29/2?

Μπορούμε να επιστρέψουμε το αντικείμενο που δημιουργούμε κατευθείαν ως επιστρεφόμενη τιμή (παρομοίως και ως όρισμα σε μέθοδο)

class Date

```
{
    private int day = 1;
    private int month = 1;
    private int year = 2014;
    private String[] monthStrings =
        {"Jan", "Feb", "Mar", "Apr", "May", "Jun",
        "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"};

    public Date(int day, int month, int year){
        this.day = day; this.month = month; this.year = year;
    }

    public String toString(){
        return day + " " + monthNames[month-1] + " " + year;
    }

    public Date nextYear(){
        if (day == 29 && month == 2){
            return null;
        }
        return new Date(day, month, year+1);
    }
}
```

Η κλάση Date

Τι γίνεται αν η ημερομηνία είναι 29/2?

Η τιμή null: Μία κενή αναφορά.
Η τιμή μπορεί να χρησιμοποιηθεί σαν μια default τιμή, ή σαν ένδειξη λάθους (στην περίπτωση αυτή ότι δεν μπορούμε να δημιουργήσουμε το αντικείμενο)

```
class DateExample
{
    public static void main(String args[]) {
        Date today = new Date(3,4,2014);
        System.out.println(today);
        Date todayNextYear = today.nextYear();
        if( todayNextYear != null) {
            System.out.println(todayNextYear);
        }
    }
}
```

Προσοχή: Η χρήση του null για έλεγχο λάθους σημαίνει ότι όποτε χρησιμοποιούμε την μέθοδο θα πρέπει να προσέχουμε αν η επιστρεφόμενη τιμή είναι null. Δεν είναι καλή λύση, και αργότερα θα μάθουμε για εξαιρέσεις για να χειριζόμαστε τέτοια προβλήματα.

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

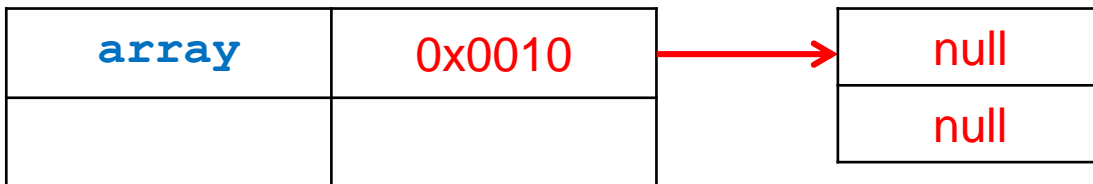
```
Person[] array;
```

<code>array</code>	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

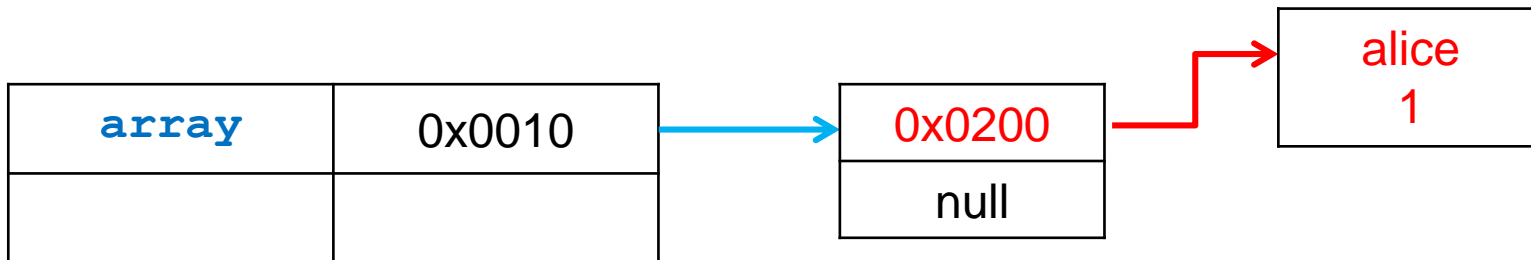
```
Person[] array;  
array = new Person[2];
```



- 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

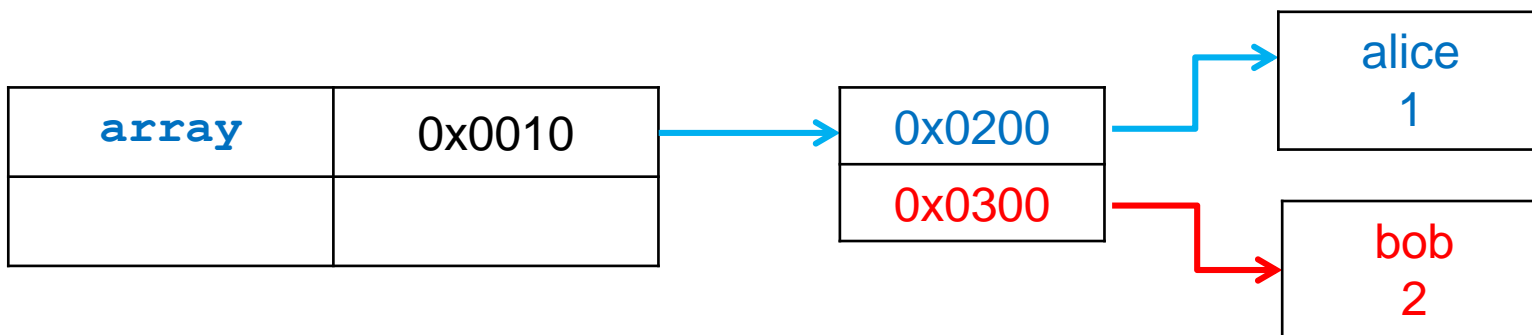
```
Person[] array;  
array = new Person[2];  
array[0] = new Person("alice", 1);
```



- 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 5×5

```
int[][] array;  
array = new int[5][];  
for (int i=0; i<5; i++){  
    array[i] = new int[5];  
}
```

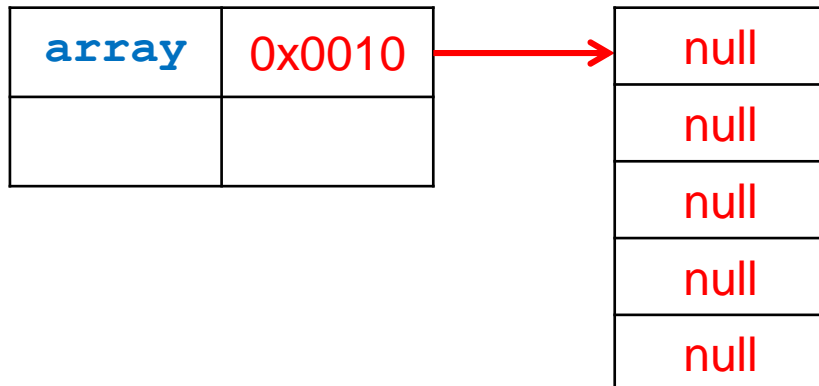
Παράδειγμα

```
int[][] array;
```

array	null

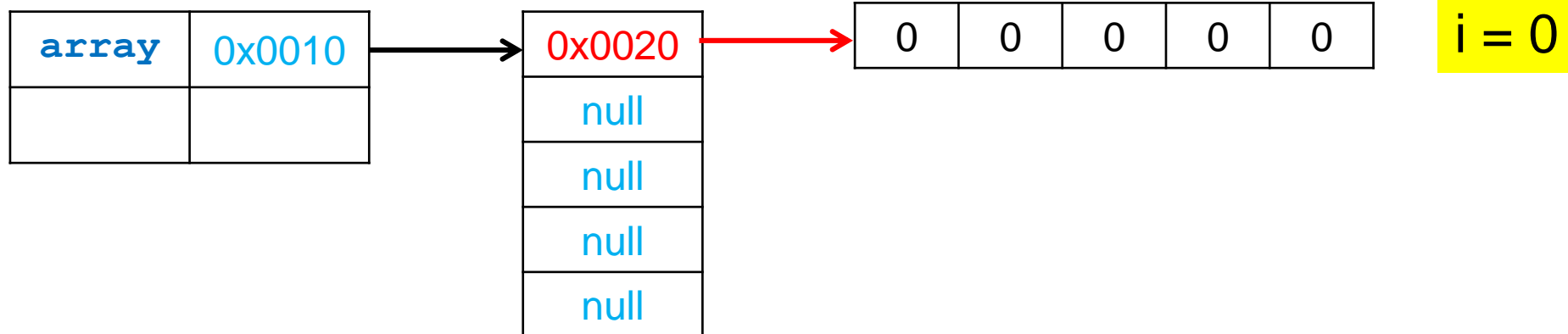
Παράδειγμα

```
int[][] array;  
array = new int[5][];
```



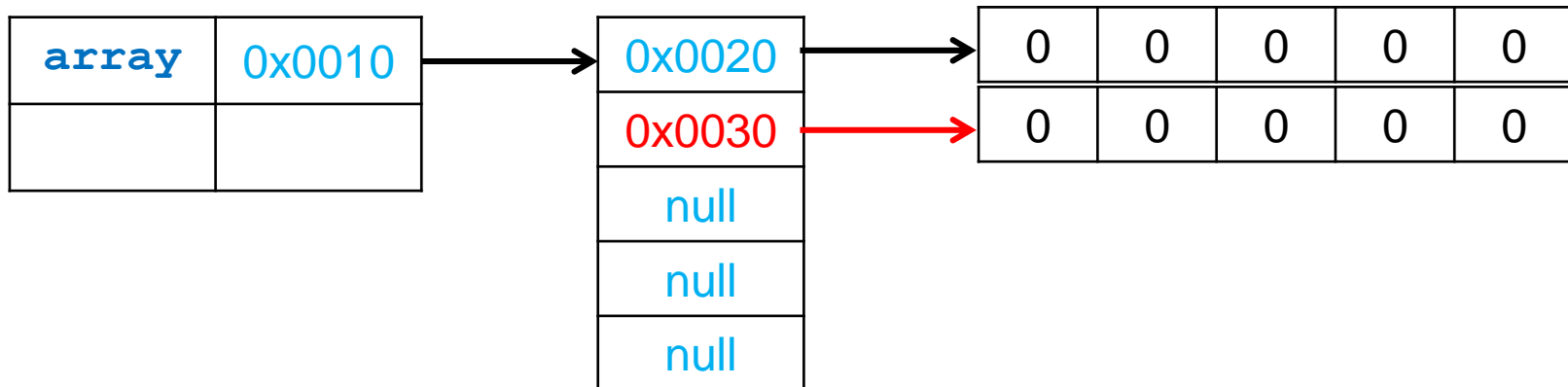
Παράδειγμα

```
int[][] array;  
array = new int[5][];  
for (int i=0; i<5; i++){  
    array[i] = new int[5];  
}
```



Παράδειγμα

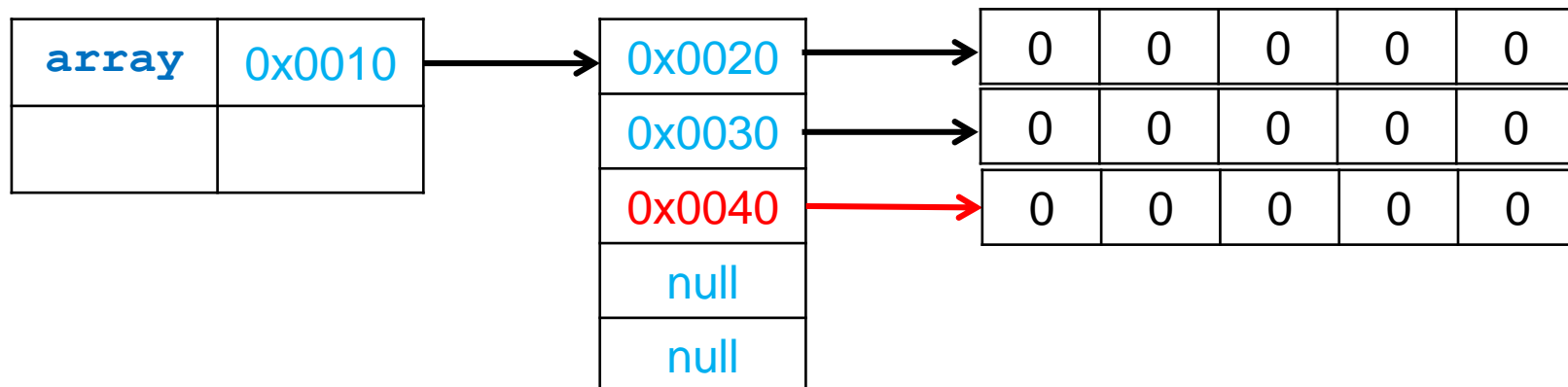
```
int[][] array;  
array = new int[5][];  
for (int i=0; i<5; i++){  
    array[i] = new int[5];  
}
```



i = 1

Παράδειγμα

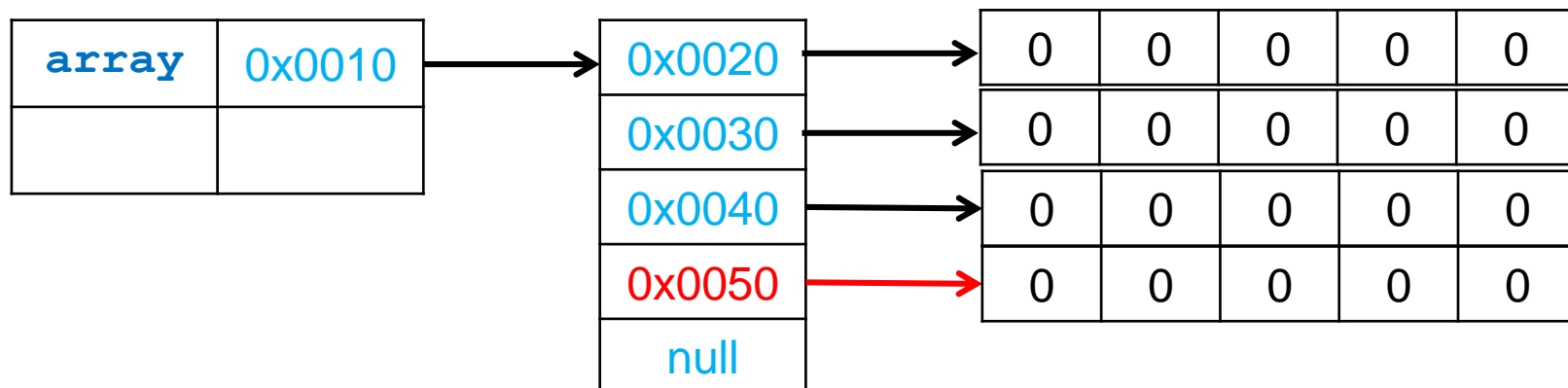
```
int[][] array;  
array = new int[5][];  
for (int i=0; i<5; i++){  
    array[i] = new int[5];  
}
```



$i = 2$

Παράδειγμα

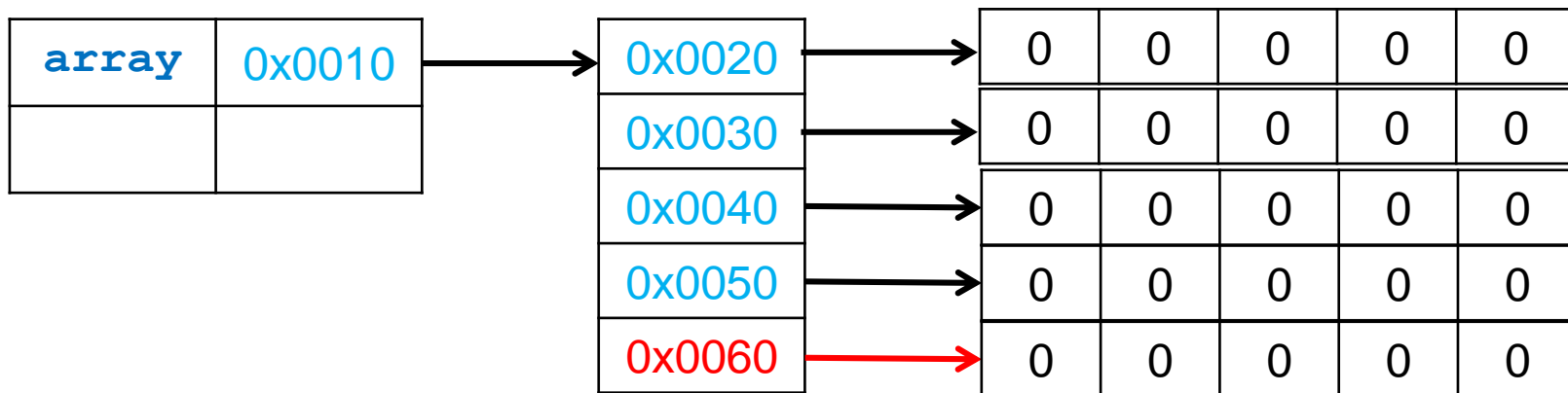
```
int[][] array;  
array = new int[5][];  
for (int i=0; i<5; i++){  
    array[i] = new int[5];  
}
```



$i = 3$

Παράδειγμα

```
int[][] array;  
array = new int[5][];  
for (int i=0; i<5; i++){  
    array[i] = new int[5];  
}
```



i = 4

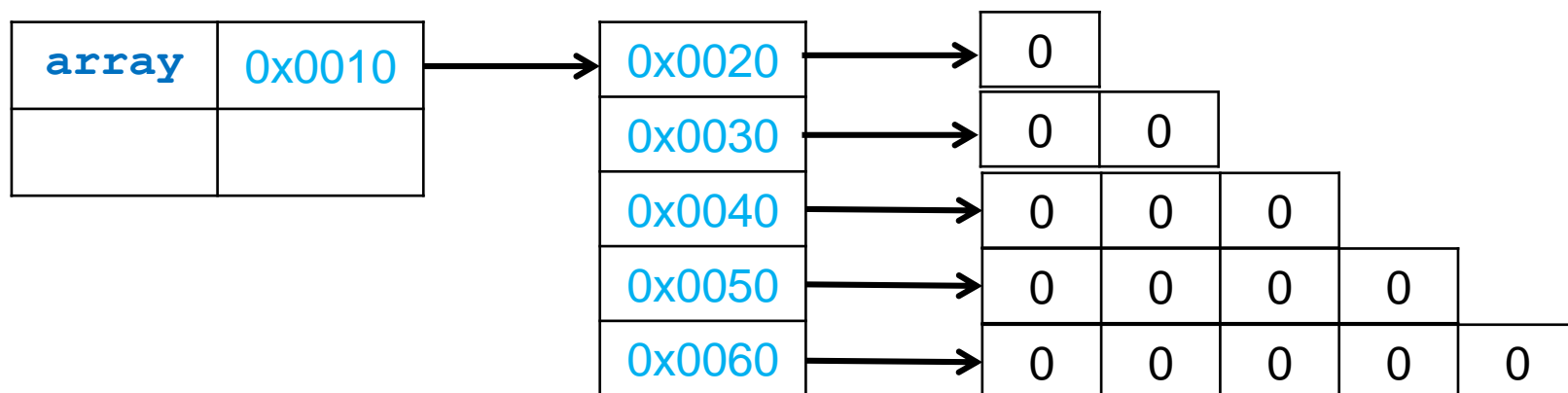
Πίνακες από πίνακες

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

```
int[][] array;  
array = new int[5][];  
for (int i=0; i<5; i++) {  
    array[i] = new int[i+1];  
}
```

Παράδειγμα

```
int[][] array;  
array = new int[5][];  
for (int i=0; i<5; i++){  
    array[i] = new int[i+1];  
}
```



```
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);
    }

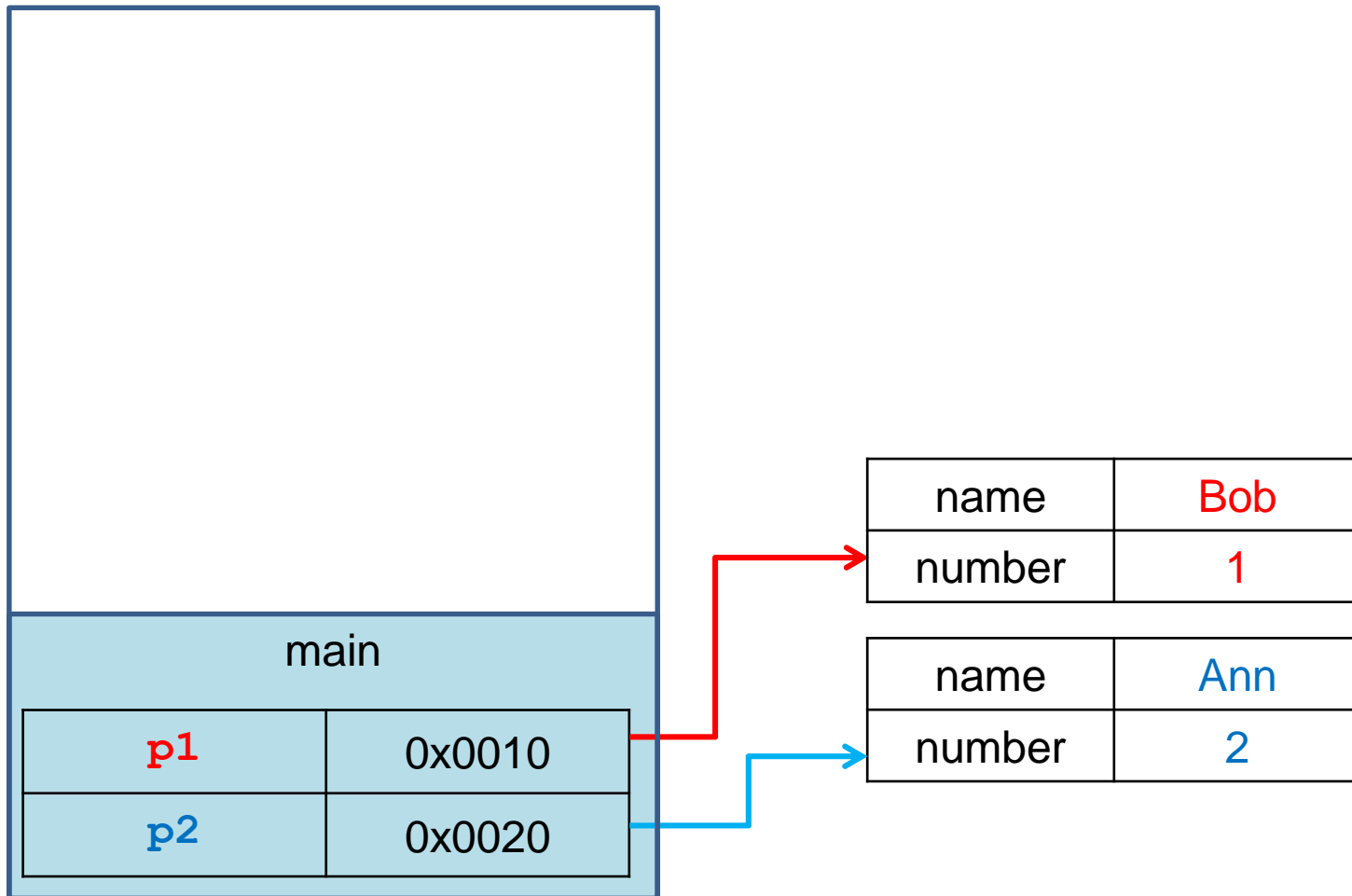
    public Person copier() {
        Person newPerson = new Person(this.name, this.number);
        return newPerson;
    }
}
```

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);
    }
}
```

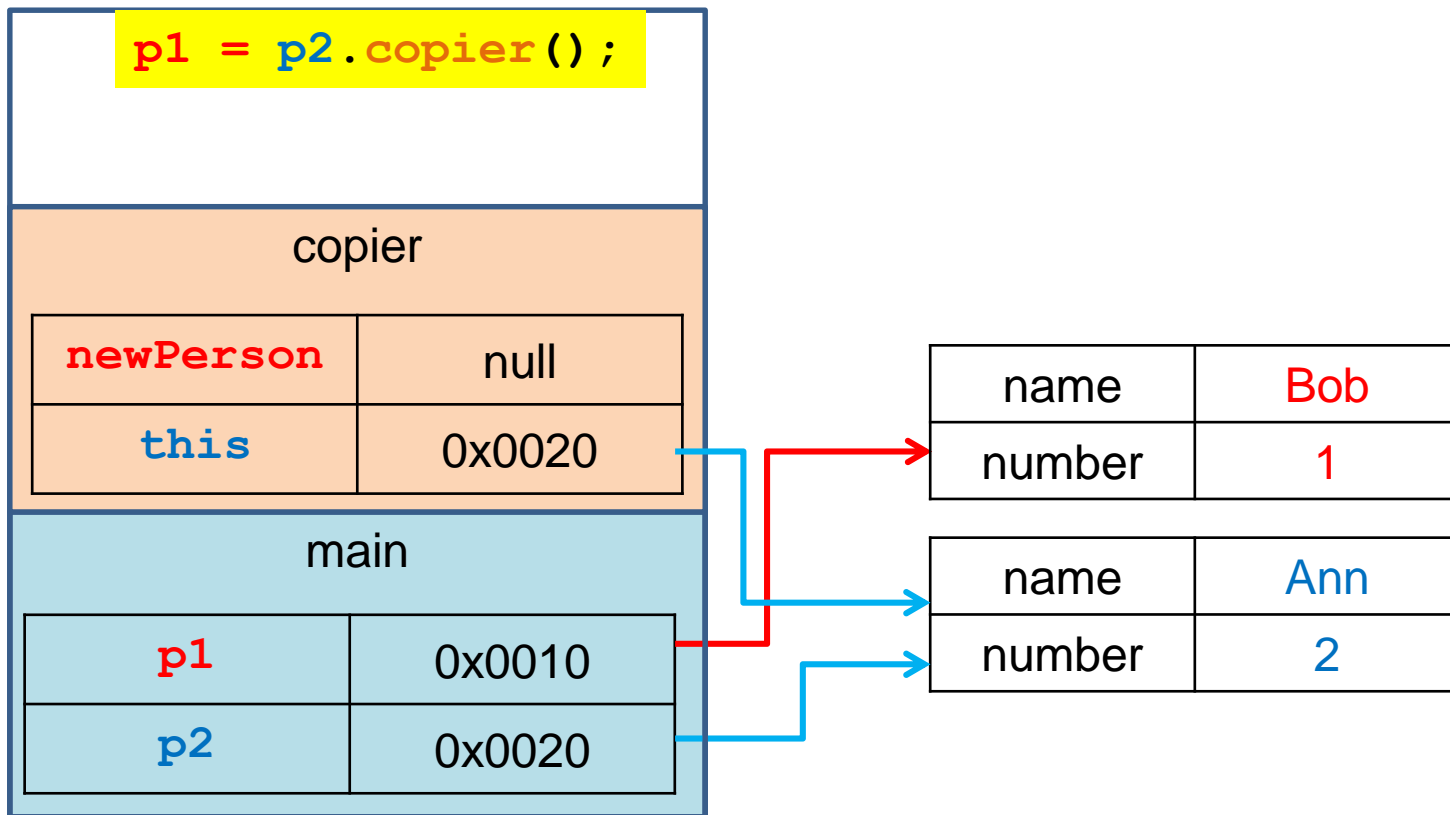
Τι θα τυπώσει?

Εξέλιξη του προγράμματος



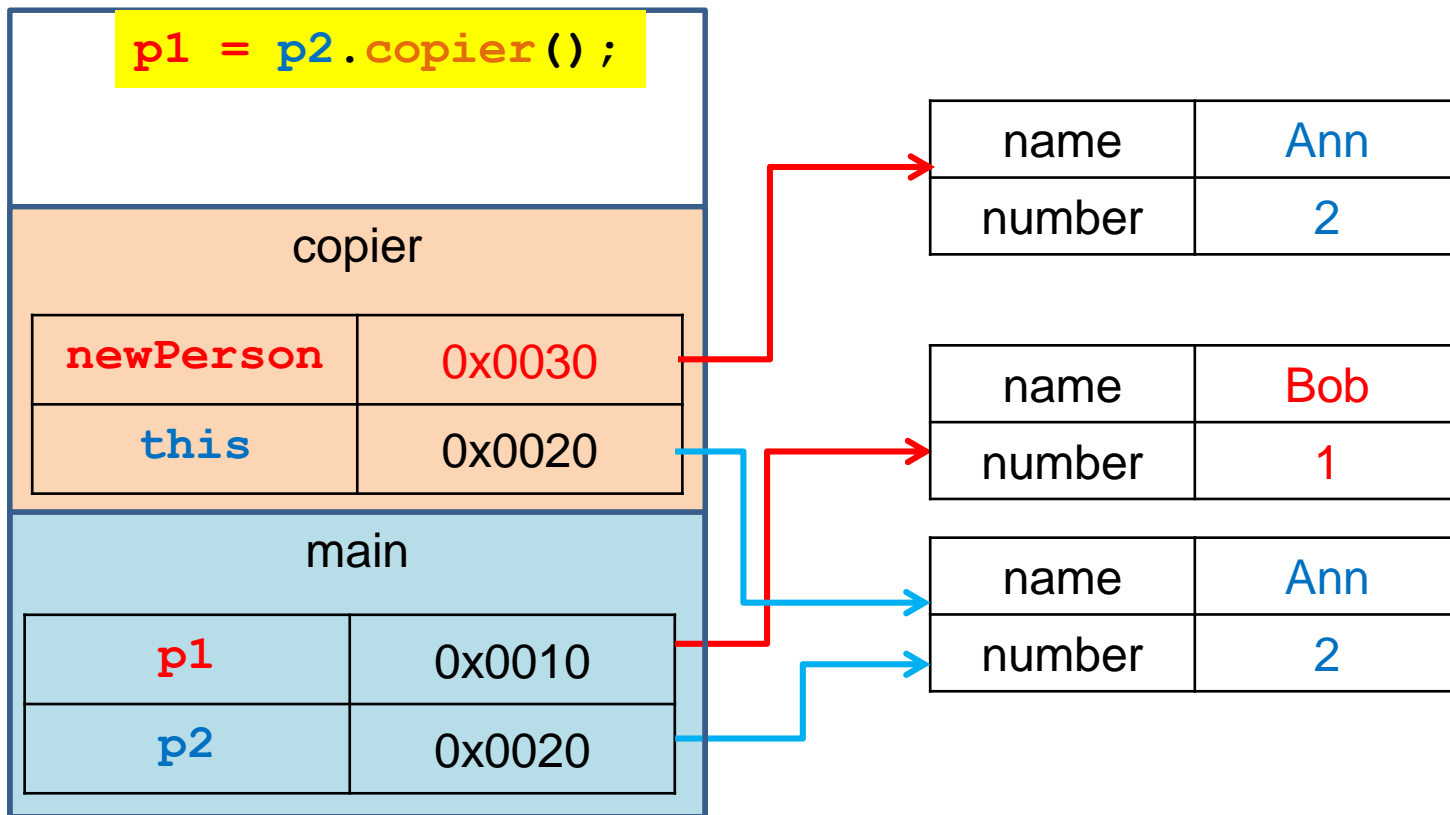
Εξέλιξη του προγράμματος

```
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;  
}
```

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

main

p1

0x0030

p2

0x0020

name

Ann

number

2

name

Bob

number

1

name

Ann

number

2

Η main τυπώνει "Ann 2"

Εξέλιξη του προγράμματος

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

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

main

p1

0x0030

p2

0x0020

name

Ann

number

2

~~name~~

~~Bob~~

~~number~~

~~1~~

name

Ann

number

2

Το προηγούμενο αντικείμενο αποδεσμεύεται

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 dim;
    private int[] position;

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

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

Το Car κινείται σε 1 ή 2 διαστάσεις
Χρειαζόμαστε ένα πίνακα για την θέση του

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

Η copy δημιουργεί και επιστρέφει ένα νέο Car

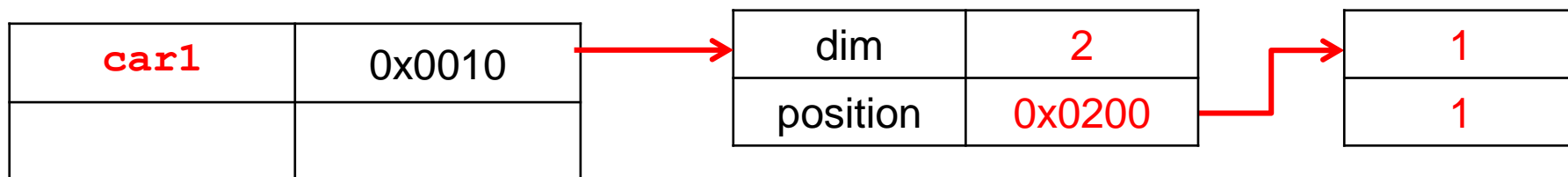
```
public String toString(){
    String output = "";
    for (int i=0; i < dim; i++){output = output + position[i] + " ";}
    return output;
}
```

```
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?

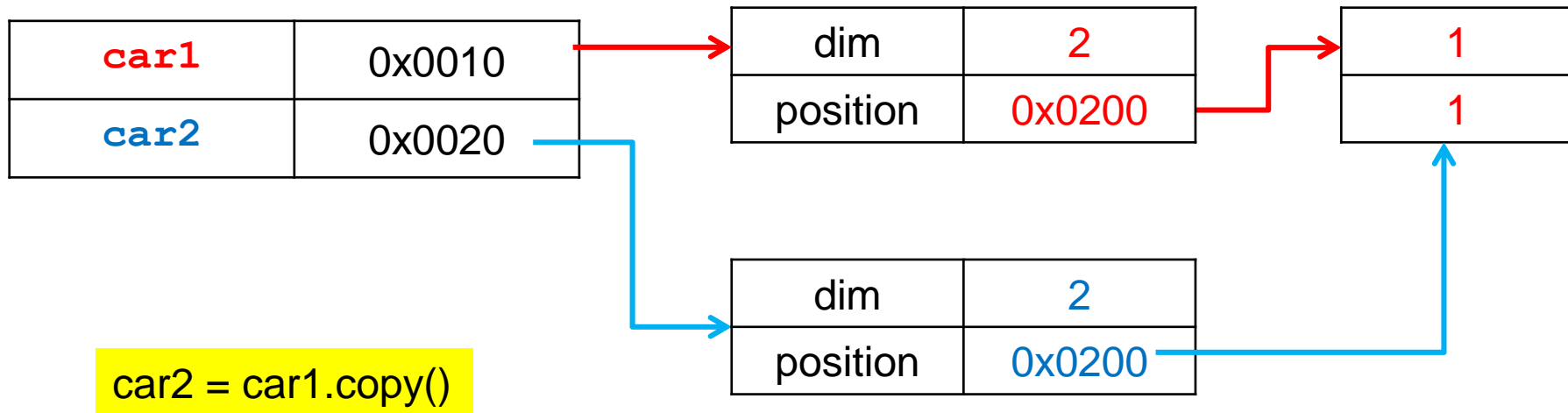
Ρηχά Αντίγραφα

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



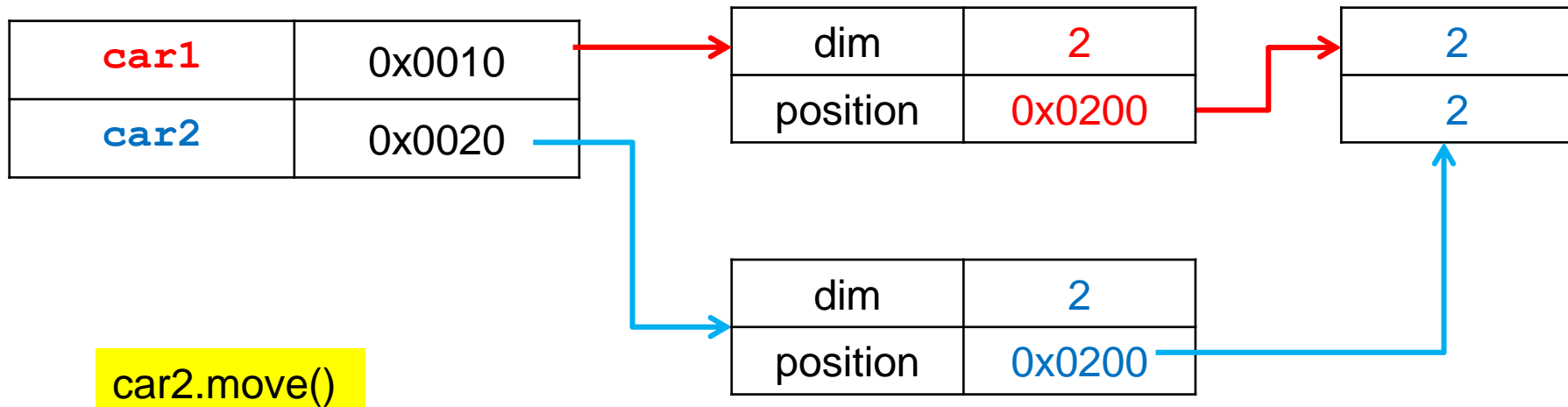
Ρηχά Αντίγραφα

- 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

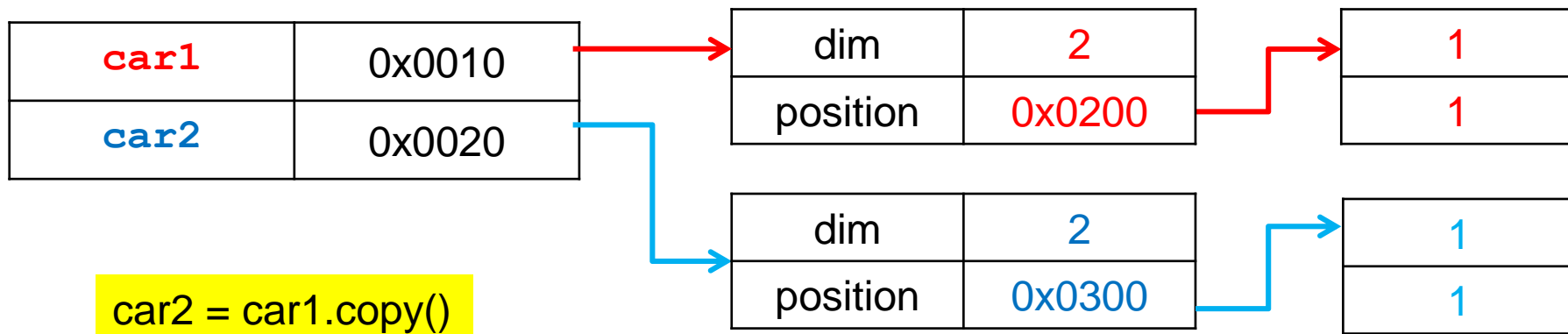


Μετακινείται και το `car1` αλλά αυτό δεν είναι επιθυμητό.

Βαθύ αντίγραφο

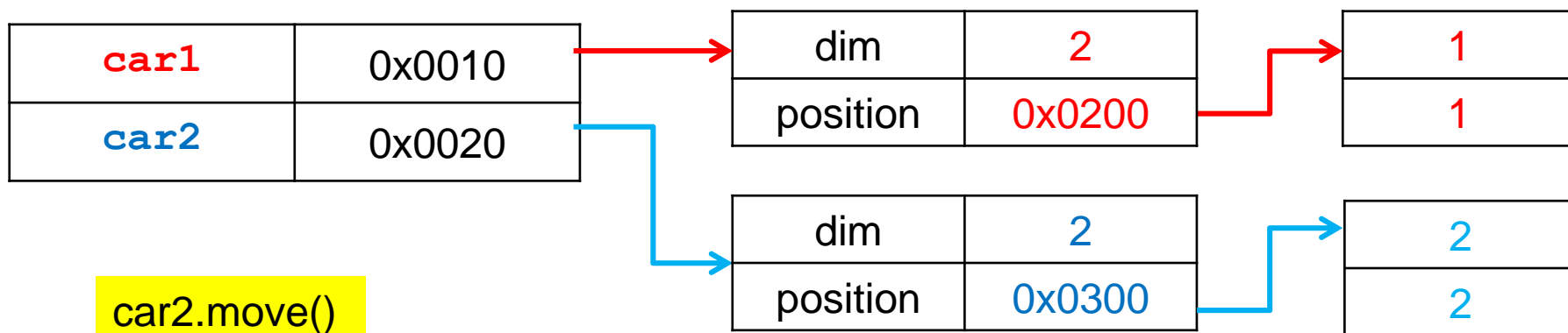
- Τις περισσότερες φορές θέλουμε να κάνουμε ένα **βαθύ αντίγραφο** του αντικειμένου, όπου για κάθε αντικείμενο μέσα στο αντίγραφο δεσμεύουμε νέα μνήμη

```
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

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**

```
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`