ΤΕΧΝΙΚΕΣ ΑΝΤΙΚΕΙΜΕΝΟΣΤΡΑΦΟΥΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ

Αναφορές
Έλεγχος ισότητας
String Interning
Αποαναφοροποίηση - dereferencing

ΕΛΕΓΧΟΣ ΙΣΟΤΗΤΑΣ

Έλεγχος ισότητας

- 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.

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

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

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

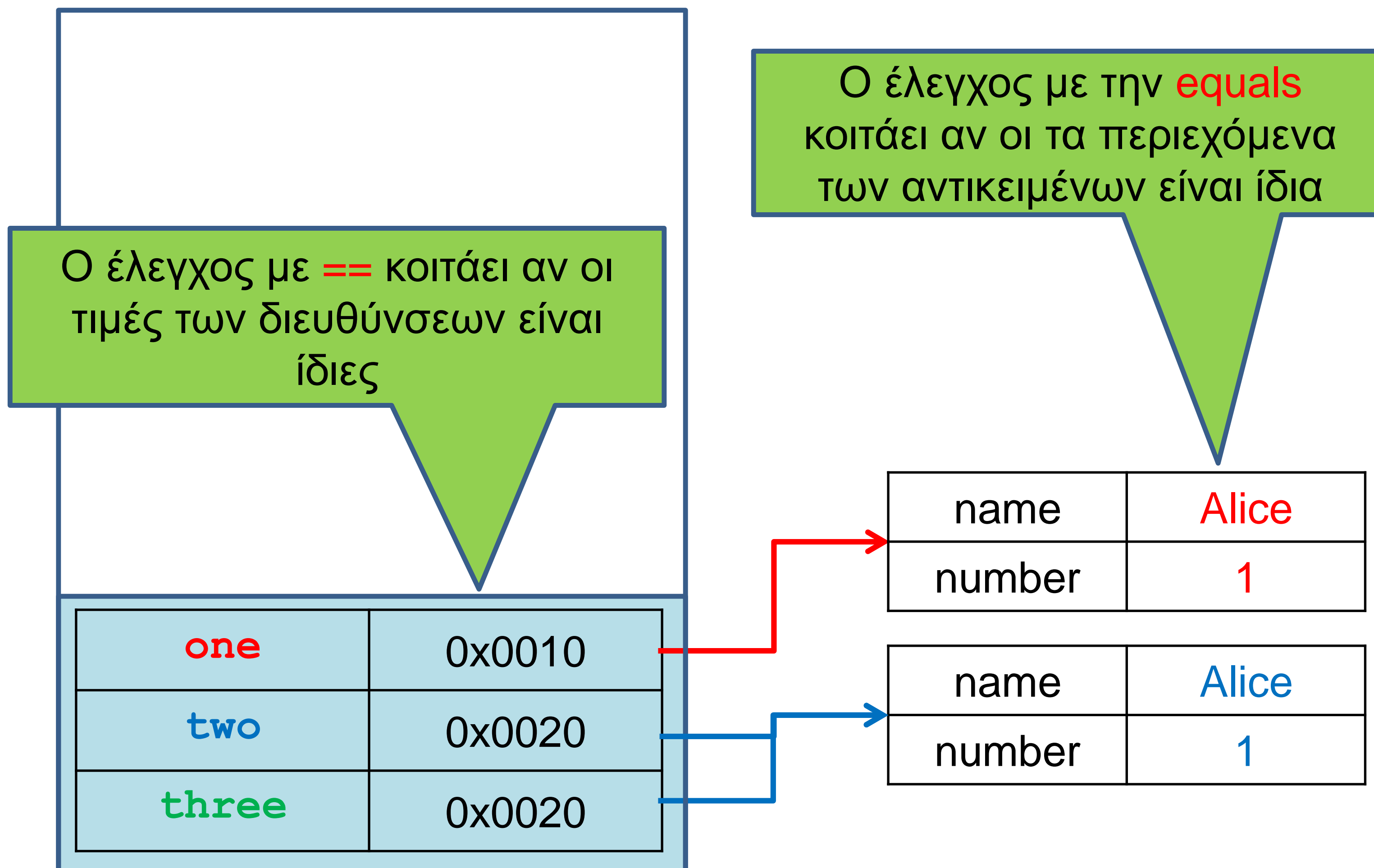
Παράδειγμα

- 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



STRING INTERNING

String Interning

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

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

Καλούμε μια μέθοδο του intern String

- Αυτό μπορεί να προκαλέσει μπερδέματα με ελέγχους ισότητας.

Ισότητα String

Τι θα εκτυπωθεί?

```
import java.util.Scanner;

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

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

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

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

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

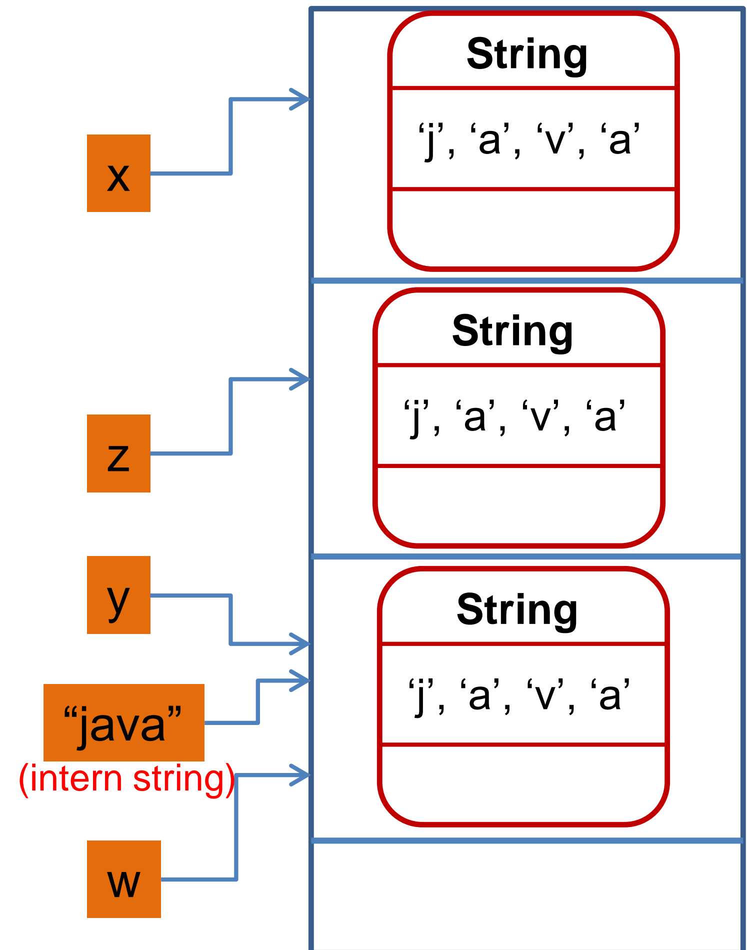
String Interning

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

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

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

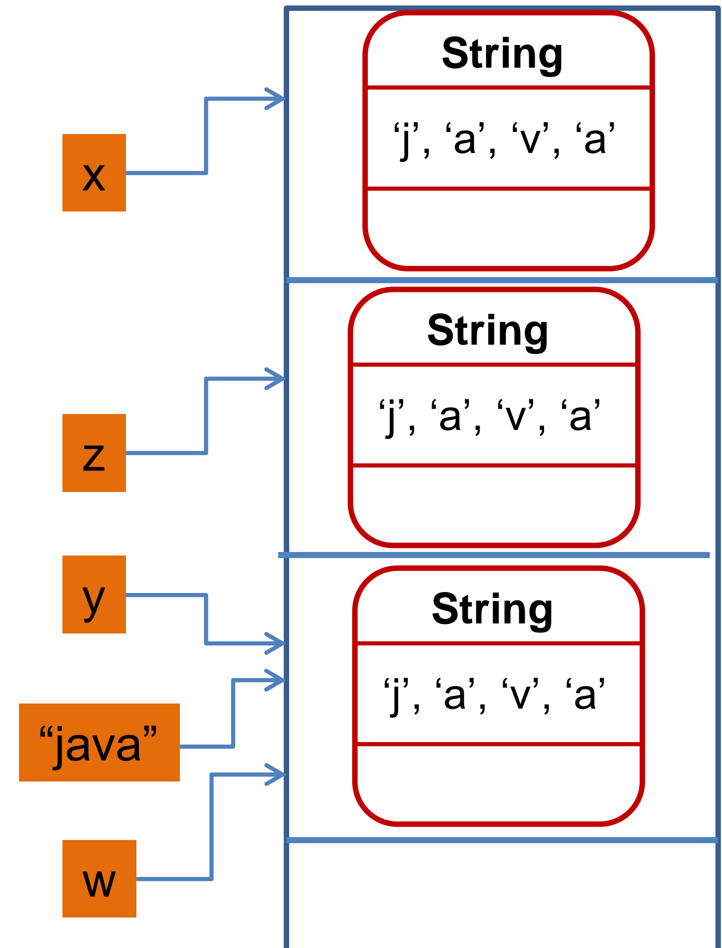
κάνει το `y` να δείχνει στη θέση που είναι αποθηκευμένη η σταθερά `"java"`



String Interning

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

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



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

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

}
```

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

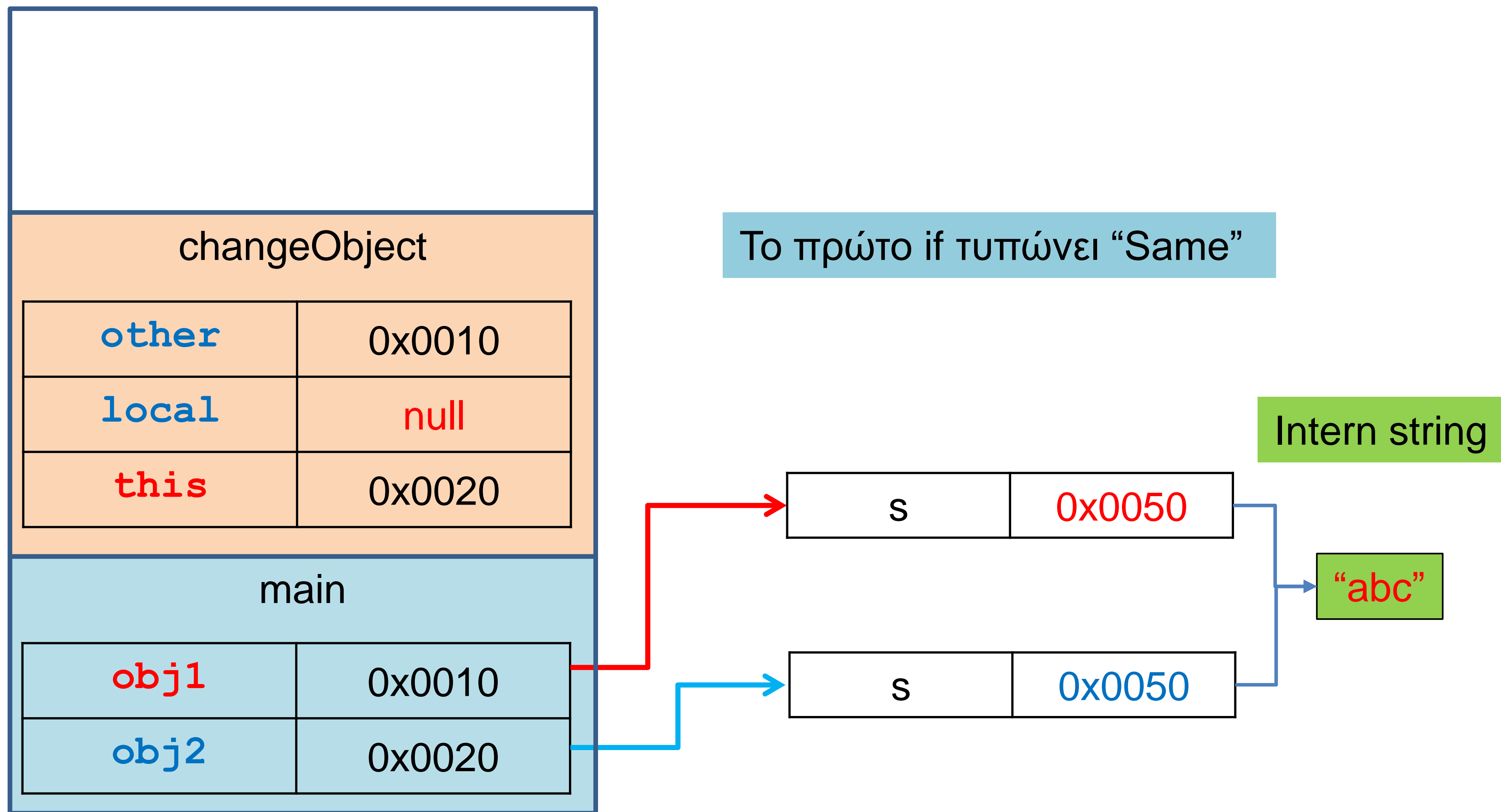
Η ανάθεση String σταθεράς είναι διαφορετική από τη δημιουργία αντικειμένου με new

Η σταθερά δημιουργεί ένα νέο intern String

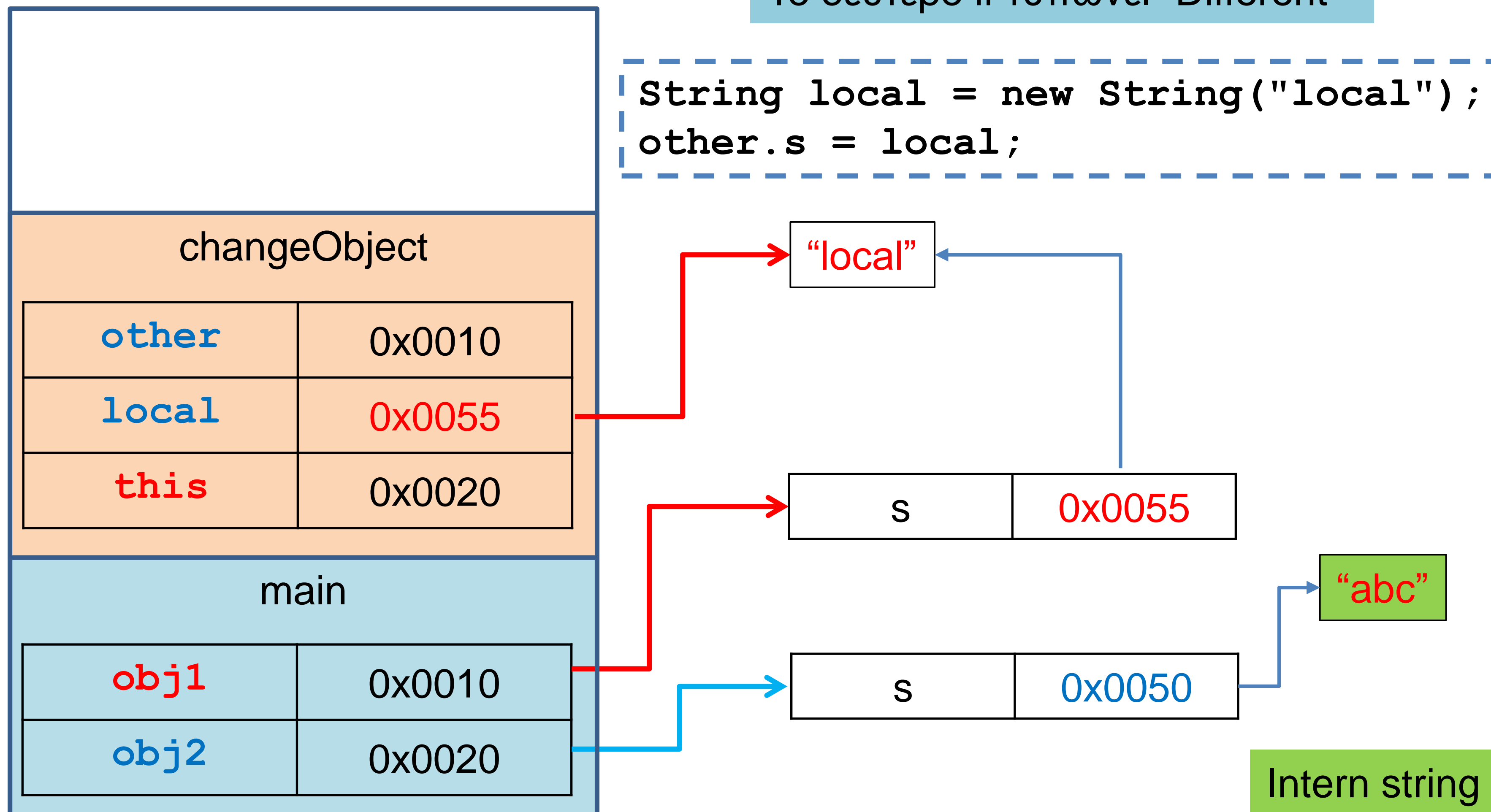
```
class StringTest{
    public static void main(String[] args){
        StringClass obj1 = new StringClass();
        StringClass obj2 = new StringClass();
        obj2.changeObject(obj1);
    }
}
```

Τι θα τυπώσει?

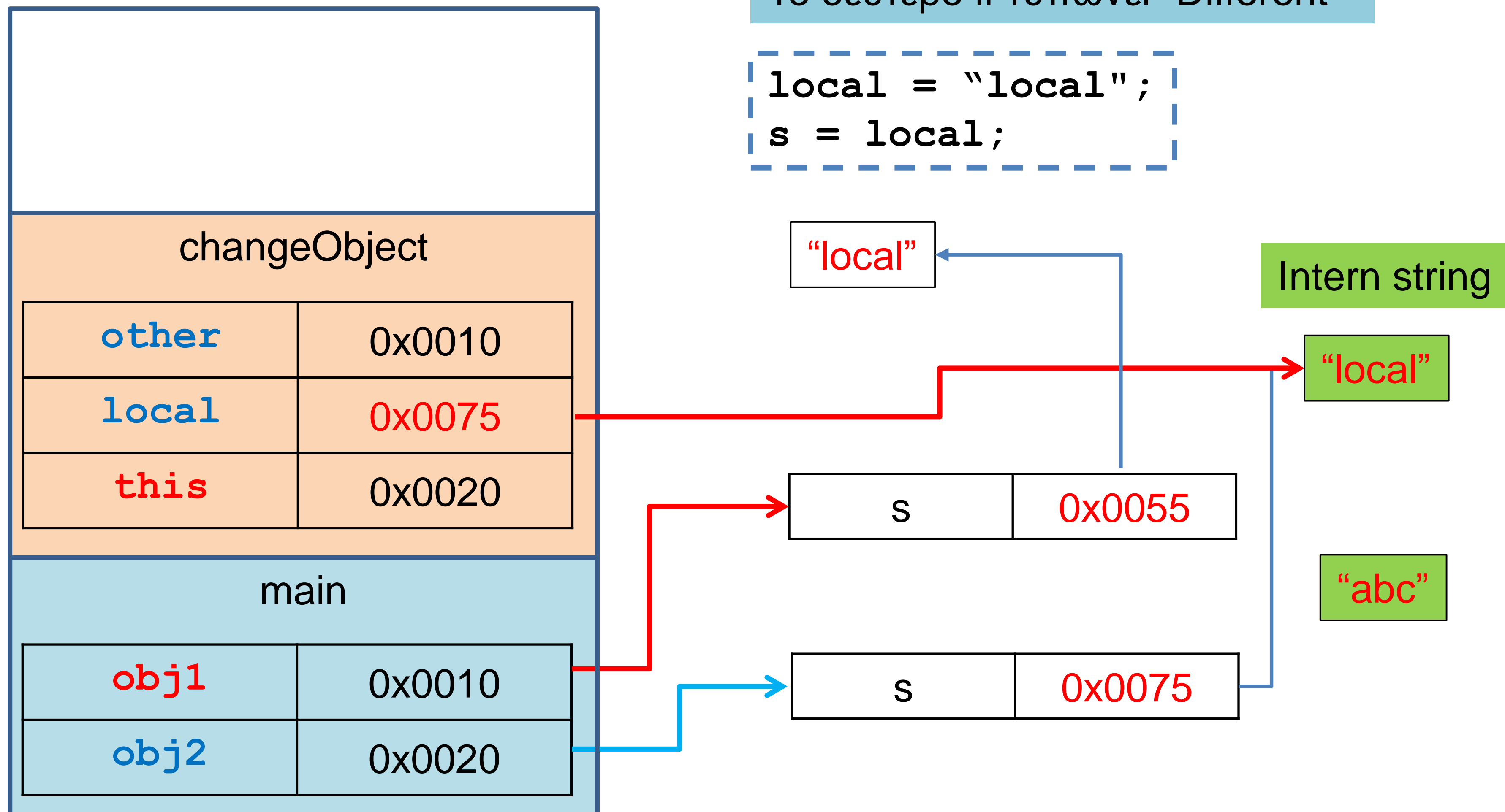
Εξέλιξη του προγράμματος



Εξέλιξη του προγράμματος



Εξέλιξη του προγράμματος



ΑΠΟ-ΑΝΑΦΟΡΟΠΟΙΗΣΗ (DEREFERENCING)

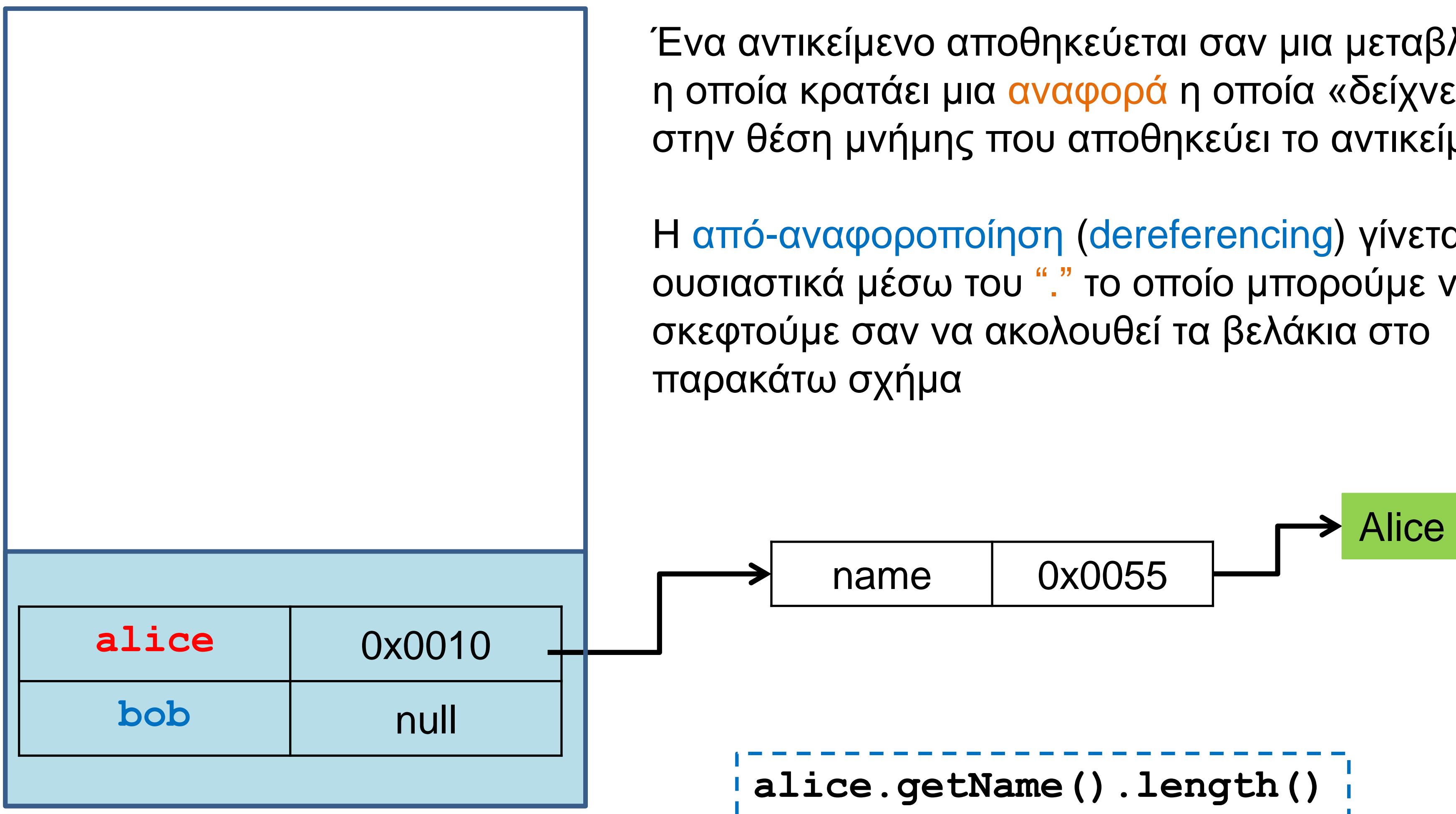
```java
class Person
{
  private String name;

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

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

```java
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.

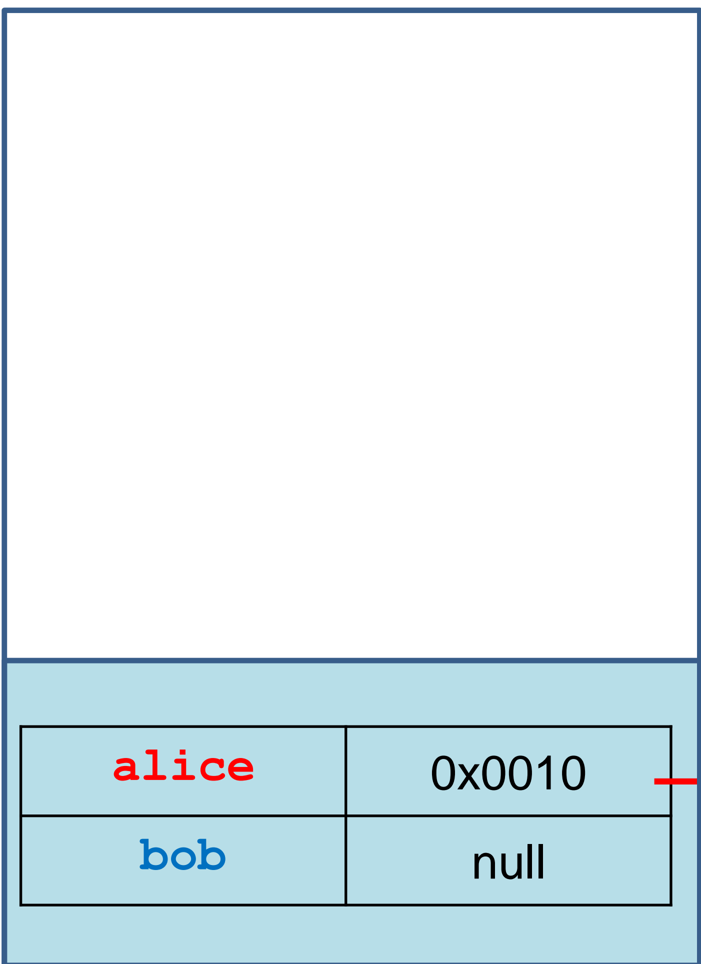
Η από-αναφοροποίηση (dereferencing) γίνεται ουσιαστικά μέσω του “.” το οποίο μπορούμε να σκεφτούμε σαν να ακολουθεί τα βελάκια στο παρακάτω σχήμα

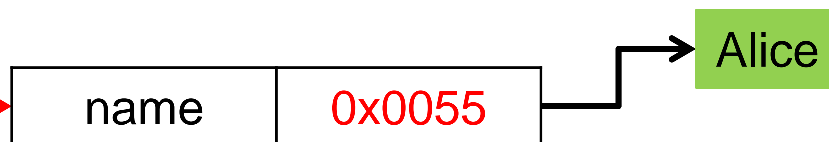


Dereferencing

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

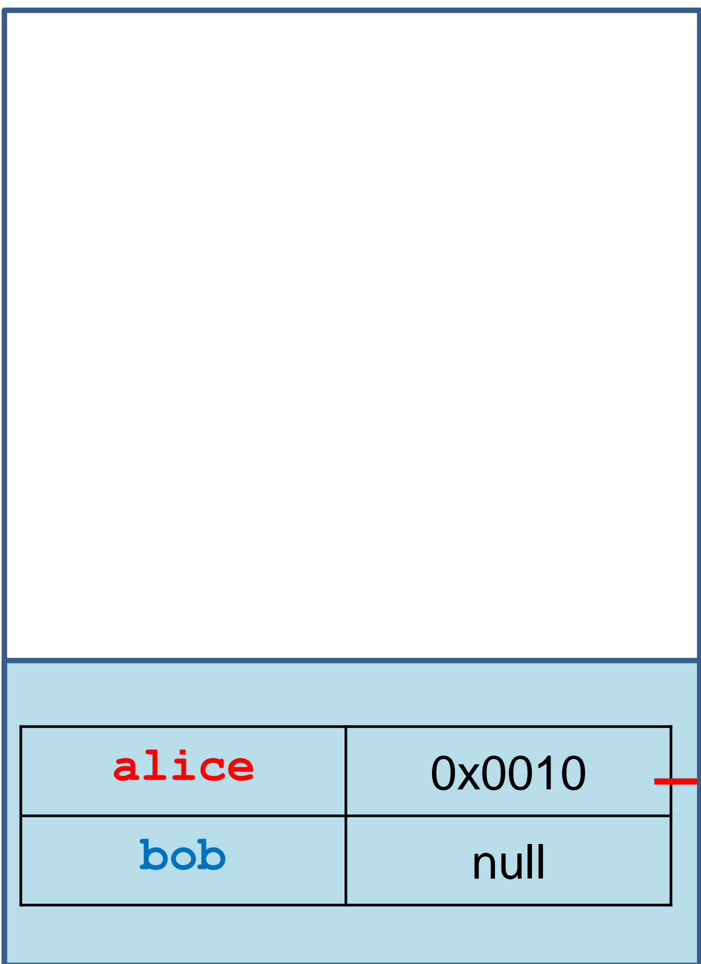
Η από-αναφοροποίηση (dereferencing) γίνεται ουσιαστικά μέσω του "." το οποίο μπορούμε να σκεφτούμε σαν να ακολουθεί τα βελάκια στο παρακάτω σχήμα





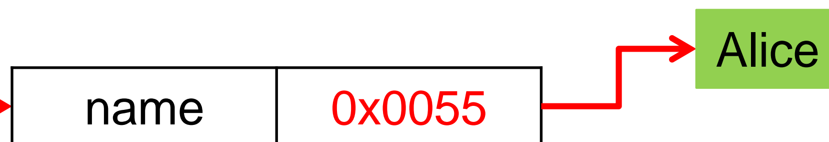

```
alice.getName().length()
```

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

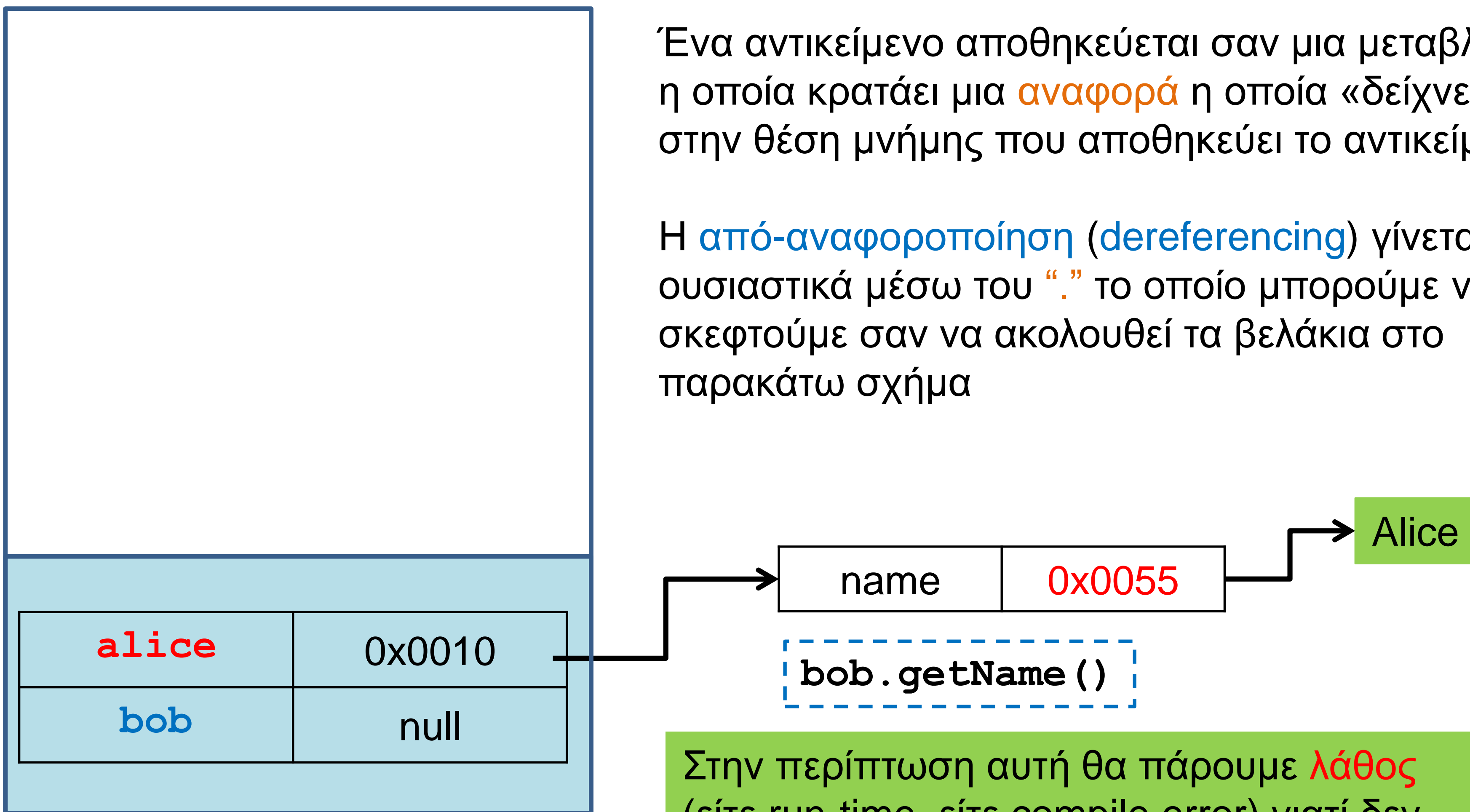



```
alice.getName().length()
```

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



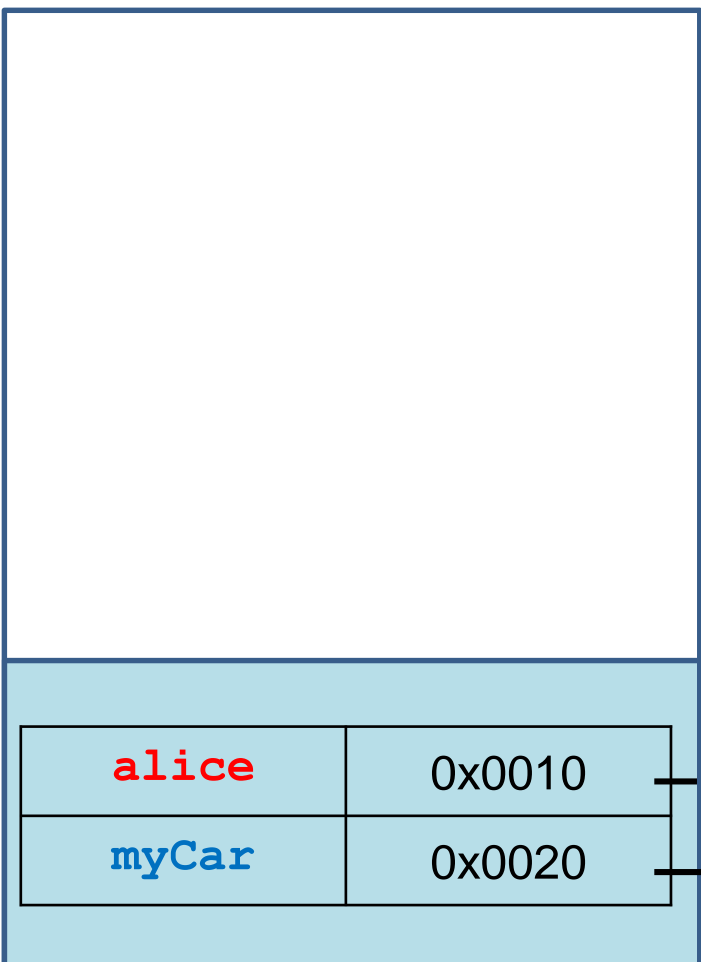
Στην περίπτωση αυτή θα πάρουμε λάθος (είτε run-time, είτε compile error) γιατί δεν υπάρχει διεύθυνση να ακολουθήσουμε

```java
class Person
{
  private String name;

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

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

```java
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;
  }
}
```

```java
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());
  }
}
```

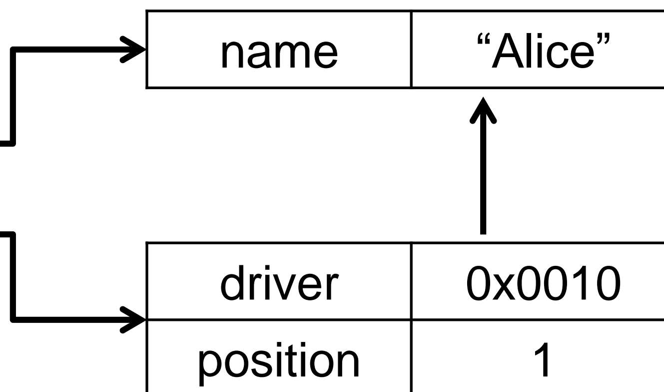
Dereferencing



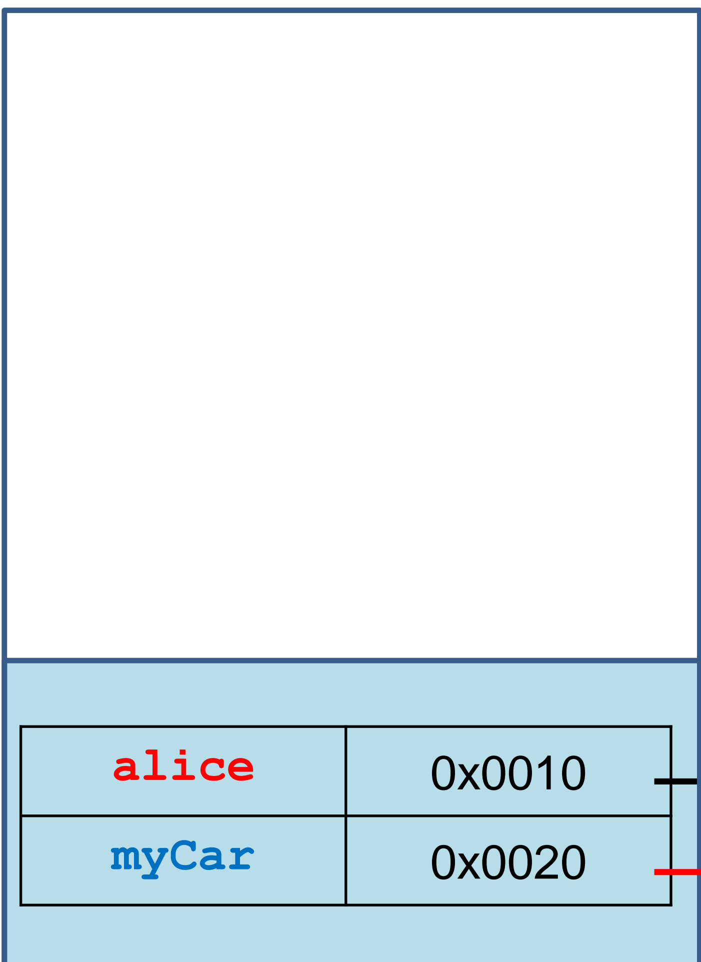
Στην περίπτωση αυτή έχουμε ένα αντικείμενο μέσα σε ένα άλλο αντικείμενο.
Η μέθοδος getDriver() επιστρέφει αντικείμενο Person

Έχουμε αλυσιδωτή πρόσβαση σε αναφορές

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



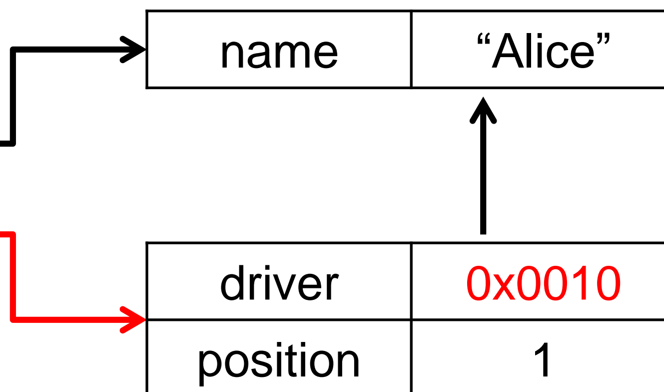
Dereferencing



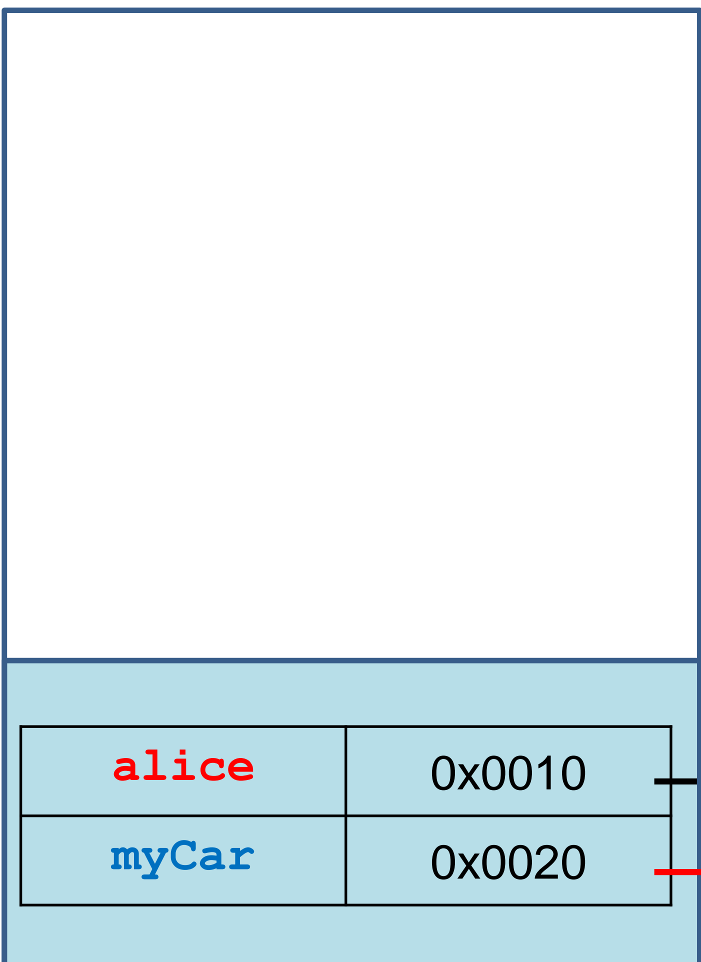
Στην περίπτωση αυτή έχουμε ένα αντικείμενο μέσα σε ένα άλλο αντικείμενο.
Η μέθοδος getDriver() επιστρέφει αντικείμενο Person

Έχουμε αλυσιδωτή πρόσβαση σε αναφορές

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



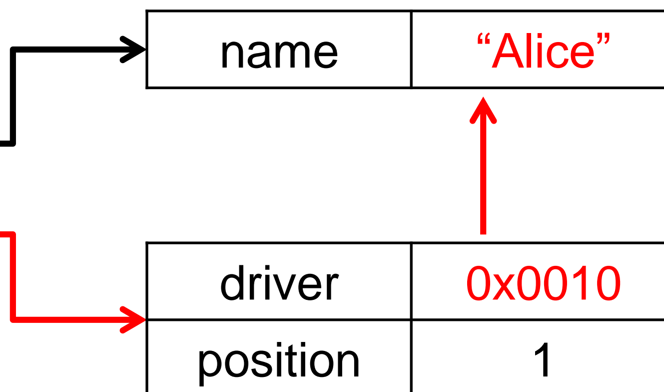
Dereferencing



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()
```

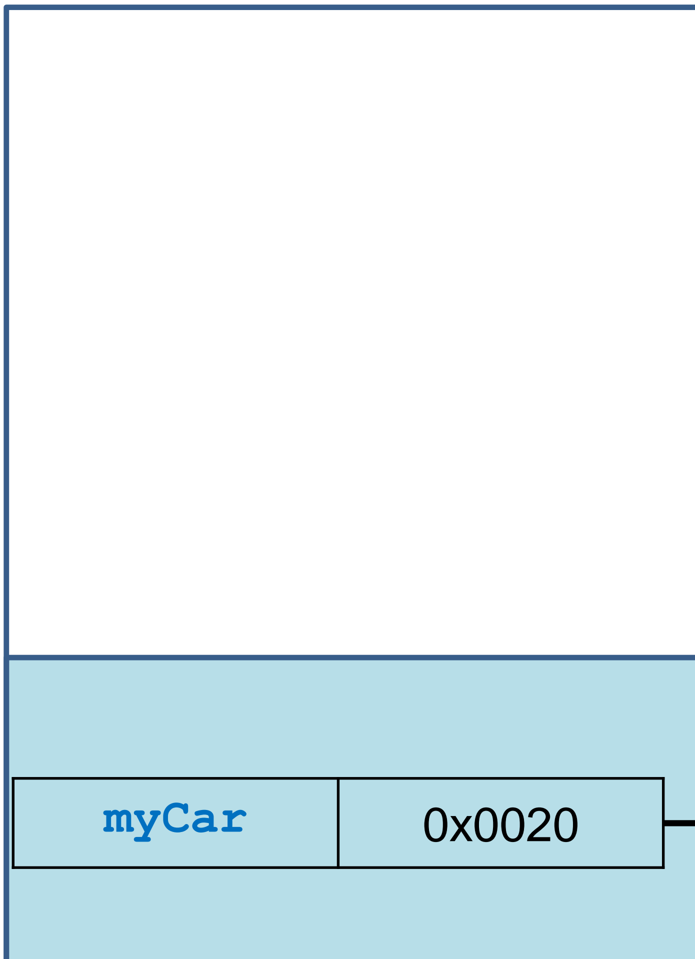


```java
class Person
{
  private String name;

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

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

```java
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();
  }
}
```

```java
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.

name	“Alice”

driver	0x0010
position	1

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.