ΤΕΧΝΙΚΕΣ ΑΝΤΙΚΕΙΜΕΝΟΣΤΡΑΦΟΥΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ

Αντικείμενα ως ορίσματα

Εισαγωγή στις αναφορές

Αντικείμενα ως ορίσματα

- Μπορούμε να περνάμε αντικείμενα ως ορίσματα σε μία μέθοδο όπως οποιαδήποτε άλλη μεταβλητή
- Οποιαδήποτε κλάση μπορεί να χρησιμοποιηθεί ως παράμετρος.
- Όταν τα ορίσματα ανήκουν στην κλάση στην οποία ορίζεται η μέθοδος τότε η μέθοδος μπορεί να δει (και) τα ιδιωτικά (private)πεδία των αντικειμένων
- Αν τα ορίσματα είναι διαφορετικού τύπου τότε η μέθοδος μπορεί μόνο να καλέσει τις public μεθόδους.

Παράδειγμα

- Η κλάση Car θα έχει ως πεδίο και το όνομα του οδηγού. Το όνομα θα το παίρνει από ένα αντικείμενο της κλάσης Person στην αρχικοποίηση.

```
class Person
{
  private String name;

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

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

```
class Car
{
  private int position = 0;
  private String driverName;

  public Car(int position, Person driver){
    this.position = position;
    driverName = driver.getName();
  }

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

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

Αντικείμενα μέσα σε αντικείμενα

- Εκτός από ορίσματα σε μεθόδους αντικείμενα οποιαδήποτε κλάσης μπορούν να εμφανιστούν και ως πεδία μιας κλάσης
  - Ένα αντικείμενο μπορεί να έχει μέσα του άλλα αντικείμενα.

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

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

Καλύτερη υλοποίηση!

```
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 toString(){
    return driver.getName()
                + " " + position;
  }
}
```

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

Το αντικείμενο δημιουργείται μέσα στον constructor
Αυτό έχει νόημα αν το Person χρησιμοποιείται μόνο μέσα στην κλάση Car

```
class Person
{
  private String name;
  private int age;

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

  public String getName(){
    return name;
  }

  public int getAge(){
    return age;
  }
}
```

```
class Car
{
  private int position = 0;
  private Person driver;

  public Car(int position, Person driver){
    this.position = position;
    if (driver.getAge() >= 18){
        this.driver = driver;
    }
  }

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

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

Η Person είναι διαφορετική κλάση άρα δεν μπορούμε να διαβάσουμε το πεδίο age

Η εντολή exit

Χρησιμοποιείται για σοβαρά λάθη για να σταματάει την εκτέλεση του προγράμματος.

```
public RandomVector(int dimension)
{
        this.position = position;
        if (driver.getAge() >= 18){
                this.driver = driver;
        }
        else{
                System.exit(-1);
        }
}
```

Αν δώσουμε αρνητική διάσταση το πρόγραμμα μας θα σταματήσει.

Το -1 εξυπηρετεί σαν κωδικός λάθους, μπορείτε να βάλετε όποια τιμή θέλετε.

```
class Person
{
  private String name;
  private int licence;

  public Person(String name,
                int licence){
      this.name = name;
      this.licence = licence;
  }
}
```

```
class Car
{
  private int position = 0;
  private Person driver;

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

Πως θα υλοποιήσουμε την toString και την equals?

```
class Person
{
  private String name;
  private int licence;

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

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

  public boolean equals(Person other){
    if (this.name.equals(other.name)&&
      this.licence == other.licence)){
        return true
    }else{
        return false;
    }
  }
}
```

```
class Car
{
  private int position = 0;
  private Person driver;

  public Car(int position, Person driver){
    this.position = position;
    this.driver = driver;
  }

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

  public boolean equals(Car other){
    if (this.position == other.position &&
        this.driver.equals(other.driver)){
        return true;
    }else{
        return false;
    }
}
```

Φωλιασμένη κλήση της toString και της equals

Κώδικας σε πολλά αρχεία

- Όταν έχουμε πολλές κλάσεις βολεύει να τις βάζουμε σε διαφορετικά αρχεία.
  - Το κάθε αρχείο έχει το όνομα της κλάσης
  - Σημείωση: μια κλάση μόνη της σε ένα αρχείο είναι by default public, μαζί με άλλη είναι by default private.
- Ένα επιπλέον πλεονέκτημα είναι ότι μπορούμε να ορίσουμε μια main συνάρτηση για κάθε κλάση ξεχωριστά
  - Βοηθάει για το testing του κώδικα.

- Για να κάνουμε compile πολλά αρχεία μαζι:
  - `javac file1.java file2.java file3.java`
    - ή μπορούμε να κάνουμε compile το “βασικό” αρχείο

Παράδειγμα

- Φτιάξετε μια κλάση που να χειρίζεται ένα λογαριασμό τράπεζας. Κρατάει το όνομα του ιδιοκτήτη και το ποσό.
- Δημιουργείστε και μία μέθοδο που συγχωνεύει δύο λογαριασμούς του ίδιου ατόμου.

```
class BankAccount
{
        private String name;
        private int amount;

        public BankAccount(String name, int amount){
                this.name = name;
                this.amount = amount;
        }

        public void merge(BankAccount other){
                if (this.name.equals(other.name)){
                        this.amount += other.amount;
                }
        }
}
```

Είναι σύνηθες το αποτέλεσμα μιας μεθόδου να αποθηκεύει το αποτέλεσμα της στο ίδιο αντικείμενο το οποίο κάλεσε την μέθοδο.

Π.χ. εδώ το αποτέλεσμα της συγχώνευσης αποθηκεύεται στον λογαριασμό που έκανε την κλήση.

```
class BankAccount
{
    private String name;
    private int amount;

    public BankAccount(String name, int amount){
        this.name = name;
        this.amount = amount;
    }

    public void merge(BankAccount other){
        if (this.name.equals(other.name)){
            this.amount += other.amount;
        }
    }

    public BankAccount mergeIntoNewAccount(BankAccount other){
        if (this.name.equals(other.name)){
            BankAccount newAccount =
                        new BankAccount(name,this.amount+other.amount);
            return newAccount;
        }
        return null;
    }
}
```

Μια άλλη επιλογή είναι να δημιουργήσουμε ένα νέο λογαριασμό μετά την συγχώνευση

Δημιουργούμε ένα νέο αντικείμενο BankAccount και το επιστρέφουμε.

Αν δεν μπορούμε να δημιουργήσουμε το νέο λογαριασμό επιστρέφουμε null. Το null είναι το κενό αντικείμενο.

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;
```

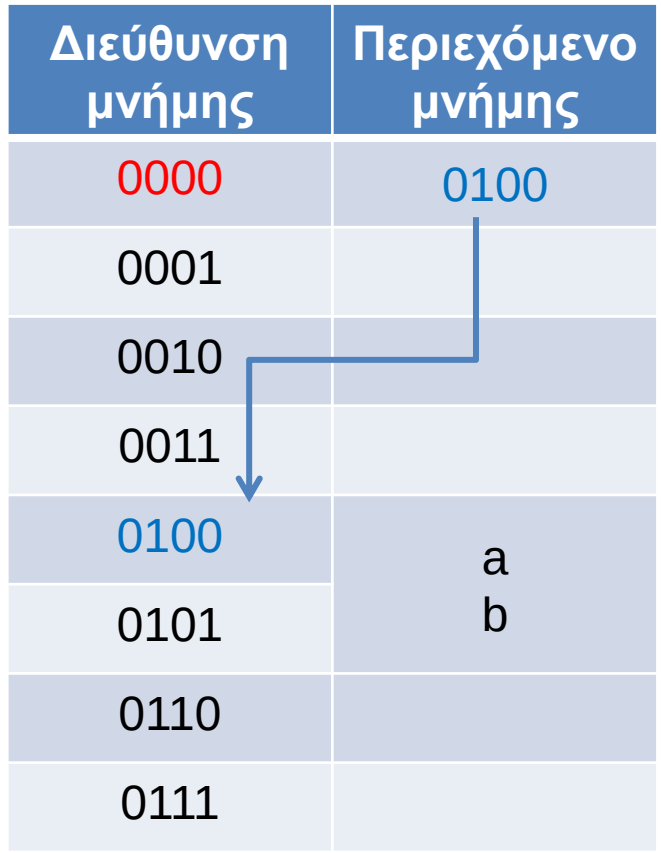
μας λέει ότι έχουμε ένα πίνακα από ακέραιους αλλά δεν μας λέει πόσο μεγάλος θα είναι αυτός ο πίνακας.

```
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];
```

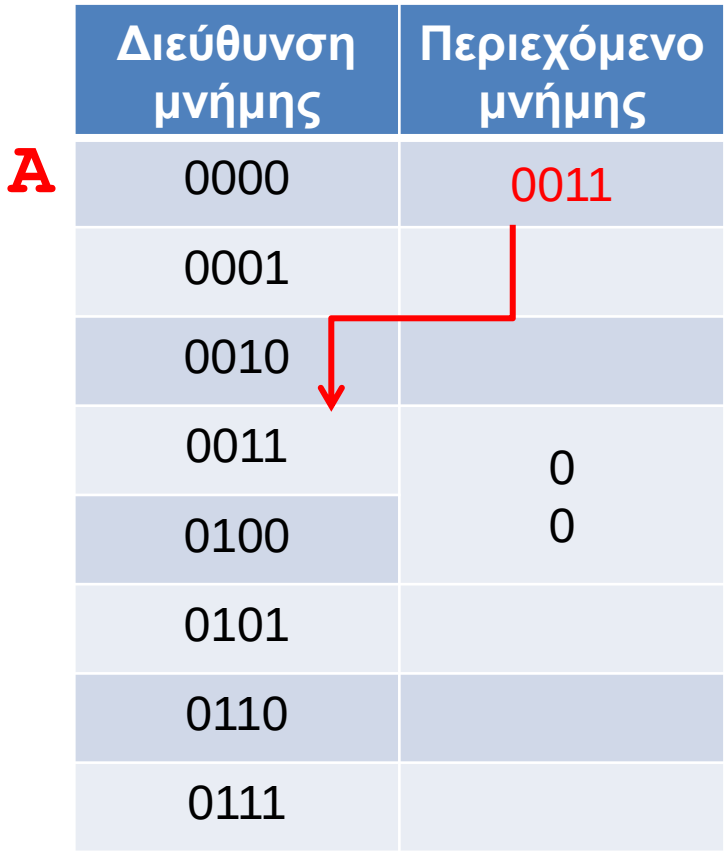
Η δεσμευμένη λέξη null σημαίνει μια κενή αναφορά (δεν δείχνει πουθενά)

	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];
```

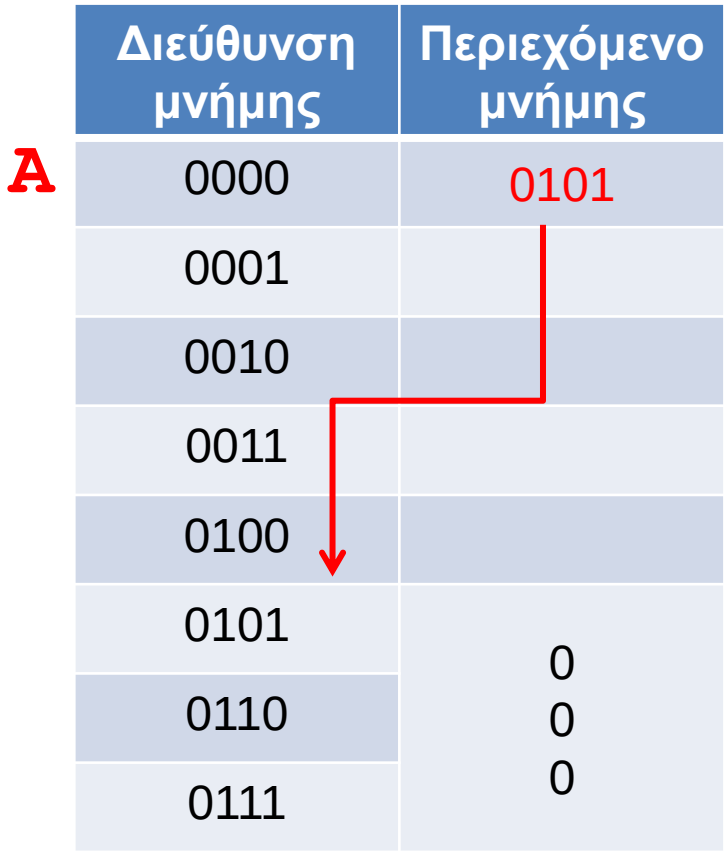
Με την εντολή new δεσμεύουμε δύο θέσεις ακεραίων και η αναφορά του A δείχνει σε αυτό το χώρο που δεσμεύσαμε



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)



Αντικείμενα κλάσεων

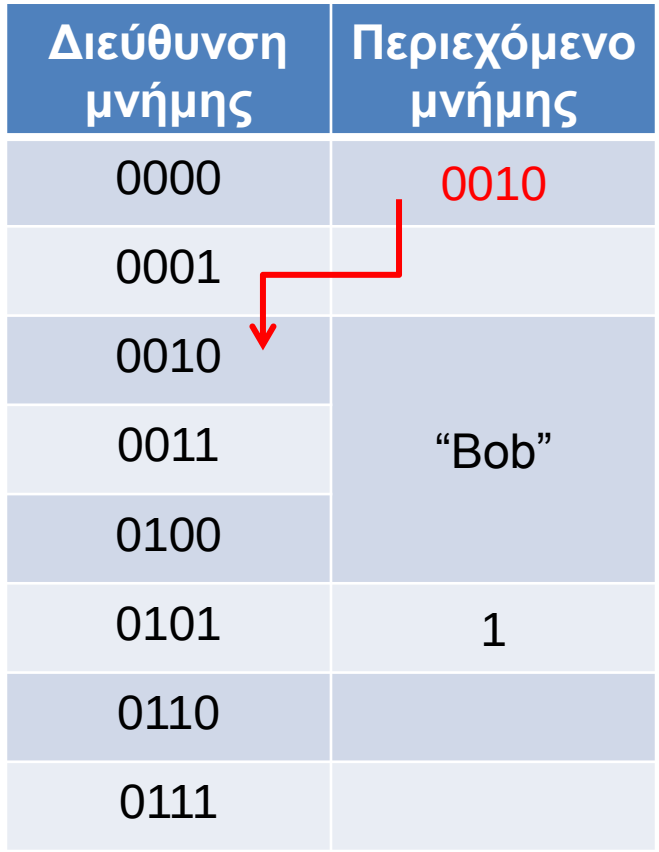
- Τι γίνεται με τα αντικείμενα κλάσεων που ορίσαμε εμείς?
- Παράδειγμα: Η κλάση Person (ToyClass από το βιβλίο).

```
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

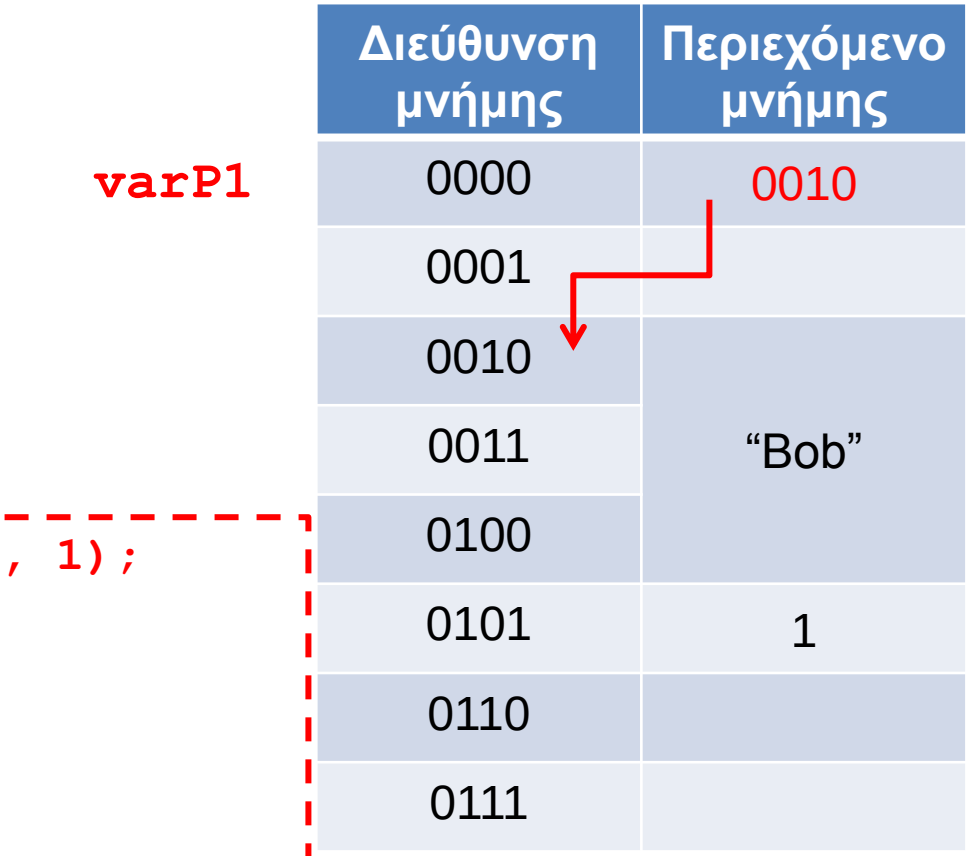
Τι θα τυπώσει το παρακάτω πρόγραμμα?

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

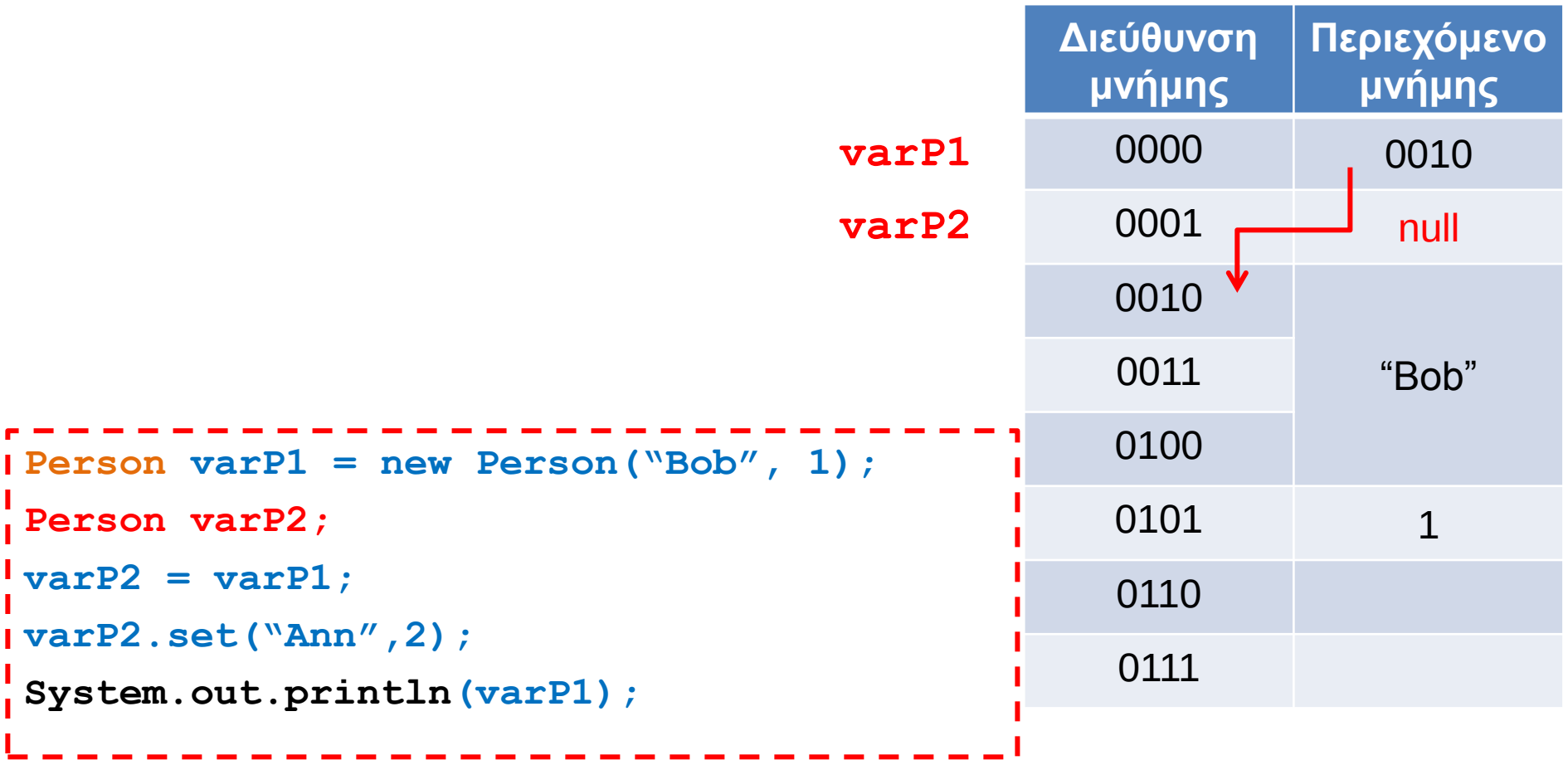
Διεύθυνση μνήμης	Περιεχόμενο μνήμης
0000	
0001	
0010	
0011	
0100	
0101	
0110	
0111	

Αναθέσεις μεταξύ αντικειμένων

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

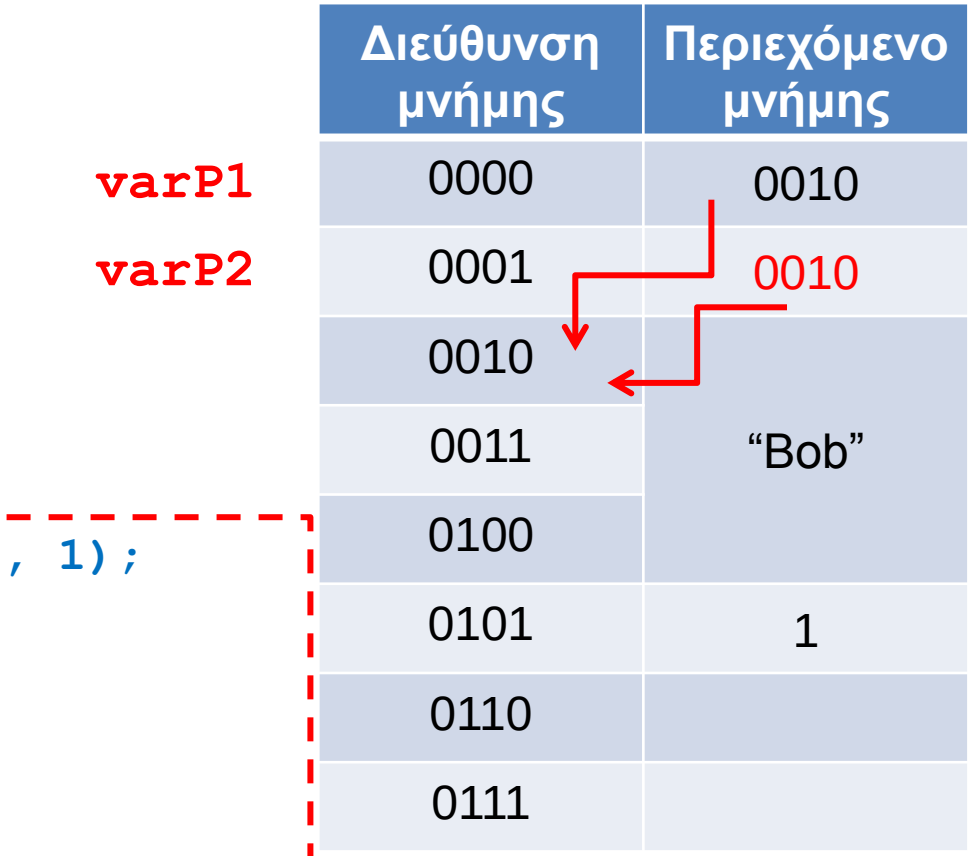


Αναθέσεις μεταξύ αντικειμένων



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	

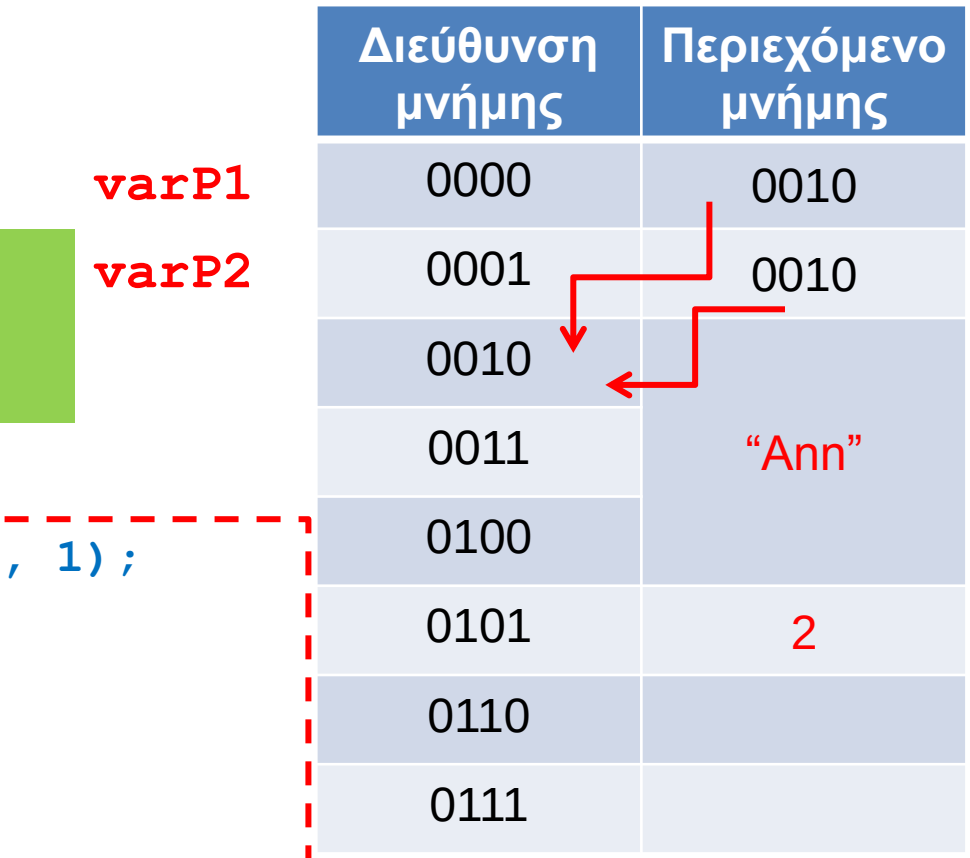
```
Person varP1 = new Person("Bob", 1);
Person varP2;
varP2 = varP1;
varP2.set("Ann",2);
System.out.println(varP1);
```

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	