Προχωρημένος Προγραμματισμός

Δημιουργία Κλάσεων και Αντικειμένων - Λεμπτομέρειες

ΕΛΕΥΘΕΡΙΟΣ ΚΟΣΜΑΣ

ΧΕΙΜΕΡΙΝΟ ΕΞΑΜΗΝΟ 2022-2023 | ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Περίληψη

Σήμερα ...

- θα συζητήσουμε τις μεθόδους δημιουργούς (constructors)
- θα συζητήσουμε την έννοια της υπερφόρτωσης (overloading)
  - θα αναφέρουμε σημεία στα οποία πρέπει να δίνουμε ιδιαίτερη σημασία όταν χρησιμοποιούμε την υπερφόρτωση
  - θα συζητήσουμε την περίπτωση της ασάφειας
- θα μελετήσουμε τη χρήση
  - αντικειμένων ως ορίσματα
  - αντικείμενα με πίνακες
  - αντικειμένων μέσα σε αντικείμενα
  - αντικειμένων ως επιστρεφόμενες τιμές
- θα συζητήσουμε τις ειδικές μεθόδους `toString` και `equals` της Java

Constructor (Δημιουργός)

- όταν δημιουργούμε ένα αντικείμενο συχνά θέλουμε να μπορούμε να το αρχικοποιήσουμε με κάποιες τιμές
  - ένα αντικείμενο `Person` να αρχικοποιείται με ένα όνομα
  - ένα αντικείμενο `Car` να αρχικοποιείται με μία θέση

- μπορούμε να το κάνουμε με μία μέθοδο `set` αυτό, αλλά
  - μπορεί να έχουμε πολλές μεταβλητές να αρχικοποιήσουμε
  - θέλουμε η αρχικοποίηση να είναι μέρος της δημιουργίας του αντικειμένου
    - ✓ εξασφαλίζουμε ότι το αντικείμενο δε χρησιμοποιείται πριν αρχικοποιηθεί

☞ την αρχικοποίηση του αντικειμένου κατά τη δημιουργία του μπορούμε να την κάνουμε με έναν Constructor (Δημιουργό)

Constructor (Δημιουργός)

- ο constructor είναι μια «μέθοδος» η οποία καλείται όταν δημιουργούμε το αντικείμενο χρησιμοποιώντας τη new

- αν δεν έχουμε ορίσει constructor καλείται ένας προεπιλεγμένος (default) constructor χωρίς ορίσματα που δεν κάνει τίποτα
  - ο προεπιλεγμένος constructor απλά εκτελεί τις αρχικοποιήσεις

- αν ορίσουμε constructor, τότε καλείται ο constructor που ορίσαμε

Constructor (Δημιουργός)

Συντακτικό

- ο constructor είναι μια μέθοδος:
  - που έχει το όνομα της κλάσης
  - ορίζεται πάντα `public`
  - δεν έχει τύπο

```
class <όνομα κλάσης>
{
        <ορισμός πεδίων>

        public <όνομα κλάσης>([ορίσματα])
        {
                [κώδικας];
        }
}
```

Constructor (Δημιουργός)

Παράδειγμα

```
1.  class Person
2.  {
3.     private String name;
4.
5.     public Person(String name) {
6.        this.name = name;
7.     }
8.
9.     public void speak(String sentence) {
10.       System.out.println(name + ": " + sentence);
11.    }
12. }
13.
27. public class HelloWorld2
28. {
29.    public static void main(String args[]) {
30.       Person alice = new Person("Alice");
31.       alice.speak("Hello World");
32.    }
33. }
```

constructor: μια μέθοδος με το ίδιο όνομα όπως και η κλάση και χωρίς τύπο (ούτε void)

αρχικοποιεί τη μεταβλητή name

constructor: καλείται όταν δημιουργείται το αντικείμενο με τη new και μόνο τότε

Constructor (Δημιουργός)

Παράδειγμα - Μια συνομιλία

```
1.  class Person {
2.     private String name;
3.
4.     public Person(String name) {
5.        this.name = name;
6.     }
7.
8.     public void speak(String sentence) {
9.        System.out.println(name + ": " + sentence);
10.    }
11. }
12.
27. public class Conversation {
28.    public static void main(String args[]) {
29.       Person alice = new Person("Alice");
30.       Person bob = new Person("Bob");
31.       alice.speak("Hi Bob");
32.       bob.speak("Hi Alice");
33.    }
34. }
```

Ενθυλάκωση

Μέθοδοι ανάγνωσης και τροποποίησης - Παράδειγμα

```
1.  class Car
2.  {
3.     private int position = 0;
4.
5.     public int getPosition() {
6.        return position;
7.     }
8.
9.     public void setPosition(int p) {
10.       position = p;
11.    }
12.
13.    public void move(int delta) {
14.       position += delta;
15.    }
16. }
```

```
17. class MovingCar
18. {
19.    public static void main(String args[])
20.    {
21.       Car myCar = new Car();
22.       myCar.setPosition(10);
23.       myCar.move(5);
24.       System.out.println("Car at position: " + myCar.getPosition());
25.    }
26. }
```

? πώς θα μπορούσαμε να υλοποιήσουμε το παράδειγμα με χρήση constructor;

Constructor (Δημιουργός)

Παράδειγμα - Αμάξι

```
1.  class Car
2.  {
3.     private int position = 0;
4.
5.     public Car(int position) {
6.        this.position = position;
7.     }
8.
9.     public void move(int delta) {
10.       position += delta;
11.    }
12.
13.    public void printPosition() {
14.       System.out.println("Car at position: " + position);
15.    }
16. }
```

```
17. public class MovingCar10 {
18.    public static void main(String args[]) {
19.       Car myCar1 = new Car(1);
20.       Car myCar2 = new Car(-1);
21.       myCar1.move(-1); myCar1.printPosition();
22.       myCar2.move(1);  myCar2.printPosition();
23.    }
24. }
```

Constructor (Δημιουργός)

Παράδειγμα - Αμάξι II

```
1.  class Car
2.  {
3.     private int position = 0;
4.     private int ACCELERATOR = 2;
5.
6.     public Car(int position) {
7.        this.position = position;
8.     }
9.
10.    public void move(int steps) {
11.       position += ACCELERATOR * steps;
12.    }
13.
14.    public void printPosition() {
15.       System.out.println("Car at position: " + position);
16.    }
17. }
```

η εκτέλεση αυτών των αρχικοποιήσεων γίνεται πριν εκτελεστούν οι εντολές στον constructor

η τελική τιμή του position θα είναι αυτή που δίνεται σαν όρισμα

```
17.  public class MovingCar11 {
18.     public static void main(String args[]) {
19.        Car myCar1 = new Car(1);
20.        Car myCar2 = new Car(-1);
21.        myCar1.move(-1); myCar1.printPosition();
22.        myCar2.move(1);  myCar2.printPosition();
23.     }
24.  }
```

Παράδειγμα - Ημερομηνίες
Περιγραφή

- θέλουμε να αναπτύξουμε μία κλάση που να αποθηκεύει ημερομηνίες
  - η κλάση θα παίρνει την ημέρα, μήνα και χρόνο σαν νούμερα
    - π.χ., 13 3 2014
  - θα μπορεί να τυπώνει την ημερομηνία με το όνομα του μήνα
    - π.χ., 13 Μαρτίου 2014

- αναπτύξτε ένα δοκιμαστικό πρόγραμμα
  - δημιουργήστε ένα αντικείμενο της κλάσης, με συγκεκριμένη ημερομηνία
  - τυπώστε την ημερομηνία

Παράδειγμα - Ημερομηνίες

Υλοποίηση

η εκτέλεση αυτών των αρχικοποιήσεων γίνεται πριν εκτελεστούν οι εντολές στον constructor

```
1.  class Date
2.  {
3.      private int day = 25;
4.      private int month = 11;
5.      private int year = 2018;
6.      private String[] monthNames = {"Jan", "Feb", "Mar", "Apr", "May", "Jun",
7.                                     "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"};
8.
9.      public Date(int day, int month, int year) {
10.         if (day < 1 ||  day > 31 || month < 1 ||  month > 12) {
11.            return;
12.         }
13.         this.day = day; this.month = month; this.year = year;
14.     }
15.
16.     public void printDate() {
17.        System.out.println(day + " " + monthNames[month-1] + " " + year);
18.     }
19. }
20.
17. public class TestDate {
18.     public static void main(String args[]) {
19.         Date myDate = new Date(30,11,2018);
20.         myDate.printDate();
21.     }
22. }
```

αν δε μπούμε στο if οι τελικές τιμές των ορισμάτων θα είναι αυτές που θα δοθούν στον constructor

αλλιώς, διατηρούνται οι αρχικές τιμές

Παράδειγμα - Ημερομηνίες

Υλοποίηση ΙΙ


```
1.    class Date
2.    {
3.        private int day = 1;
4.        private int month = 1;
5.        private int year = 2018;
6.        private String[] monthNames = {"Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"};
7.
8.        public Date(int day, int month, int year) {
9.            if (checkDay(day)) { this.day = day; }
10.           if (checkMonth(month)) { this.month = month; }
11.           this.year = year;
12.       }
13.
14.       private boolean checkDay(int day) {
15.           if (day < 1 ||  day > 31) { return false; }
16.           return true;
17.       }
18.
19.       private boolean checkMonth(int month) {
20.           if (month < 1 ||  month > 12) { return false; }
21.           return true;
22.       }
23.
24.       public void printDate() {
25.          System.out.println(day + " " + monthNames[month-1] + " " + year);
26.       }
27.   }
```

ο constructor μπορεί να καλεί και άλλες μεθόδους που κάνουν κάποια από τη δουλειά που χρειάζεται

Παράδειγμα - Φοιτητής

Περιγραφή

- Θέλουμε να αναπτύξουμε μία κλάση (`Student`) που να αποθηκεύει πληροφορίες για έναν φοιτητή
  - τι πεδία πρέπει να έχουμε;
  - τι θα μπει στον constructor;

Παράδειγμα - Φοιτητής

Υλοποίηση

```
1.  class Student
2.  {
3.     private String name = "John Doe";
4.     private int AM = 1000;
5.
6.     public Student(String name, int AM) {
7.        this.name = name;
8.        this.AM = AM;
9.     }
10.
11.    public void printInfo() {
12.       System.out.println(name + " " + AM);
13.    }
14.
17.    public static void main(String args[]) {
18.       Student aStudent = new Student("Giwrgos", 2001);
19.       myDate.printInfo();
20.    }
21. }
```

Παράδειγμα - Λίστα επισκεπτών

Περιγραφή

- Θέλουμε να αναπτύξουμε μία κλάση (`GuestList`) που να χειρίζεται τους καλεσμένους σε ένα πάρτυ
  - τι πεδία πρέπει να έχουμε;
  - τι θα μπει στον constructor;

Παράδειγμα - Λίστα επισκεπτών

Υλοποίηση


```
1.  class GuestList
2.  {
3.     private String[] names;
4.     private boolean[] confirm;
5.     private int numberOfGuests;
6.
7.     public GuestList(int numberOfGuests) {
8.        this.numberOfGuests = numberOfGuests;
9.        names = new String[numberOfGuests];
10.       confirm = new boolean[numberOfGuests];
11.
12.       /*  εδώ μπορούμε να έχουμε κώδικα για τις τιμές
13.        *  ή να εισάγονται τα ονόματα ένα ένα
14.        *  ή μπορεί η εισαγωγή ονομάτων να γίνει με άλλες μεθόδους της κλάσης
15.        */
16.    }
17. }
```

ορίζει τους πίνακες με τα ονόματα των καλεσμένων και τις επιβεβαιώσεις, αλλά δεν δεσμεύει χώρο γιατί δεν ξέρουμε τον αριθμό των προσκεκλημένων

αρχικοποίηση του αριθμού των επισκεπτών

Δεσμεύει μνήμη για τους πίνακες με τα ονόματα των καλεσμένων και τις επιβεβαιώσεις

Παράδειγμα - Λίστα επισκεπτών

Υλοποίηση ΙΙ - με μέθοδο για προσθήκη προσκεκλημένων


```
1.  class GuestList
2.  {
3.      private String[] names;
4.      private boolean[] confirm;
5.      private int numberOfGuests;
6.      private int guestsSoFar = 0;
7.
8.      public GuestList(int numberOfGuests) {
9.          this.numberOfGuests = numberOfGuests;
10.         names = new String[numberOfGuests];
11.         confirm = new boolean[numberOfGuests];
12.     }
13.
14.     public void addGuest(String name, boolean confirmation) {
15.         if (guestsSoFar == numberOfGuests) { return; }
16.         names[guestsSoFar] = name;
17.         confirm[guestsSoFar] = confirmation;
18.         guestsSoFar++;
19.     }
20. }
```

χρειαζόμαστε αυτή τη μεταβλητή για να ξέρουμε πόσους επισκέπτες έχουμε προσθέσει μέχρι τώρα

αν έχει γεμίσει η λίστα μας δεν προσθέτουμε

η **guestsSoFar** μας δίνει και την επόμενη άδεια θέση στον πίνακα

Η κλάση Car

- μία κλάση που κρατάει τη θέση ενός αυτοκινήτου
  - `move()`: μετακινεί το αυτοκίνητο κατά μία θέση προς τα μπρος
  - `moveManySteps(int steps)`: μετακινεί `steps` θέσεις (μπρος ή πίσω)

- και οι δύο μέθοδοι ουσιαστικά υλοποιούν το `move`
  - με τη διαφορά ότι η μία παίρνει όρισμα και η άλλη όχι
  - θα ήταν καλύτερα να μπορούσαμε να χρησιμοποιήσουμε το ίδιο όνομα και για τις δύο μεθόδους

- η Java μας δίνει αυτή τη δυνατότητα μέσω της διαδικασίας της υπερφόρτωσης (overloading)

Υπερφόρτωση (Overloading)

- η Java μας δίνει τη δυνατότητα να ορίσουμε πολλές μεθόδους με το ίδιο όνομα μέσω της διαδικασίας της υπερφόρτωσης (overloading)
  - ορισμός πολλών μεθόδων με το ίδιο όνομα
  - αλλά διαφορετικά ορίσματα,
  - μέσα στην ίδια κλάση

- για να μπορεί να γίνει σωστά η υπερφόρτωση θα πρέπει οι μέθοδοι να έχουν διαφορετική υπογραφή
  - η υπογραφή μίας μεθόδου είναι το όνομα της και η λίστα με τους τύπους των ορισμάτων της μεθόδου

- η Java μπορεί να ξεχωρίσει μεθόδους με διαφορετική υπογραφή

Υπερφόρτωση (Overloading)

Παράδειγμα: Αμάξι

```
1.  class Car
2.  {
3.     private int position = 0;
4.
5.     public Car(int position) {
6.        this.position = position;
7.     }
8.
9.     public void move() {
10.       position ++;
11.    }
12.
13.    public void move(int steps) {
14.       position += steps;
15.    }
16. }
```

οι μέθοδοι
- **move()**
- **move(int)**

έχουν **διαφορετική** υπογραφή

```
17. public class MovingCar12 {
18.    public static void main(String args[]) {
19.       Car myCar = new Car(1);
20.       myCar.move();
21.       myCar.move(-1);
22.    }
23. }
```

μετακινεί το όχημα μια θέση μπροστά

μετακινεί το όχημα μια θέση πίσω

Υπερφόρτωση (Overloading)
Δημιουργών

- είναι αρκετά συνηθισμένο να υπερφορτώνουμε τους δημιουργούς των κλάσεων

Υπερφόρτωση (Overloading)

Δημιουργών - Παράδειγμα

```
1.  class Car
2.  {
3.     private int position;
4.
5.     public Car() {
6.        this.position = 0;
7.     }
8.
9.     public Car(int position) {
10.       this.position = position;
11.    }
12.
13.    public void move() {
14.       position ++;
15.    }
16.
17.    public void move(int steps) {
18.       position += steps;
19.    }
20. }
```

```
17. public class MovingCar13 {
18.    public static void main(String args[]) {
19.       Car myCar1 = new Car(1);
20.       myCar1.move();
21.       Car myCar2 = new Car();
22.       myCar2.move(-1);
23.    }
24. }
```

Υπερφόρτωση (Overloading)

Δημιουργών - Παράδειγμα II

```
1.  class Car
2.  {
3.     private int position = 0;
4.
5.     public Car() { }
6.
7.     public Car(int position) {
8.        this.position = position;
9.     }
10.
11.    public void move() {
12.       position ++;
13.    }
14.
15.    public void move(int steps) {
16.       position += steps;
17.    }
18. }
```

κενός κώδικας, χρειάζεται για να οριστεί ο προεπιλεγμένος δημιουργός

γενικά είναι καλό να ορίζετε και ένα δημιουργό χωρίς ορίσματα

```
17. public class MovingCar14 {
18.    public static void main(String args[]) {
19.       Car myCar1 = new Car(1);
20.       myCar1.move();
21.       Car myCar2 = new Car();
22.       myCar2.move(-1);
23.    }
24. }
```

Υπερφόρτωση (Overloading)
Προσοχή!

- όταν ορίζουμε ένα δημιουργό, ο προεπιλεγμένος δημιουργός παύει να υπάρχει!
  - πρέπει να τον ορίσουμε μόνοι μας

Υπερφόρτωση (Overloading)

Προσοχή - Παράδειγμα


```
1.  class Car
2.  {
3.     private int position = 0;
4.
5.     public Car(int position) {
6.        this.position = position;
7.     }
8.
9.     public void move() {
10.       position ++;
11.    }
12.
13.    public void move(int steps) {
14.       position += steps;
15.    }
16. }
```

```
17. public class MovingCar15 {
18.    public static void main(String args[]) {
19.       Car myCar1 = new Car(1);
20.       myCar1.move();
21.       Car myCar2 = new Car();
22.       myCar2.move(-1);
23.    }
24. }
```

Θα χτυπήσει λάθος ότι δεν υπάρχει δημιουργός χωρίς ορίσματα

Υπερφόρτωση (Overloading)
Προσοχή! II

- η υπερφόρτωση γίνεται μόνο ως προς τα ορίσματα
  - όχι ως προς την επιστρεφόμενη τιμή

- η υπογραφή μίας μεθόδου είναι το όνομα της και η λίστα με τους τύπους των ορισμάτων της μεθόδου
  - η Java μπορεί να ξεχωρίσει μεθόδους με διαφορετική υπογραφή
  - π.χ., `move()` και `move(int)` έχουν διαφορετική υπογραφή

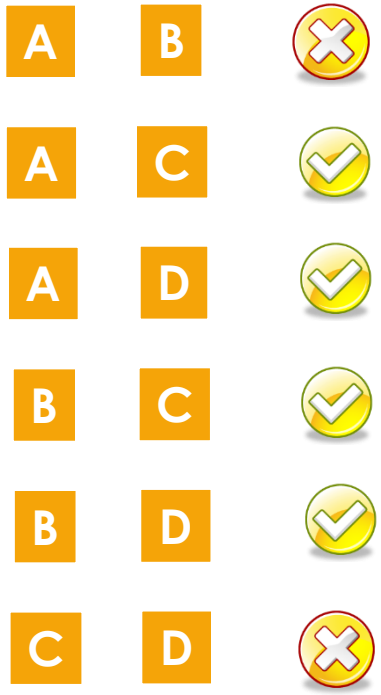
☞ όταν δημιουργούμε μία μέθοδο θα πρέπει να δημιουργούμε μία διαφορετική υπογραφή

Υπερφόρτωση (Overloading)

Προσοχή! ΙΙ - Παράδειγμα


```
1.  class SomeClass
2.  {
3.      public int aMethod(int x, double y) {
4.  A       System.out.println("int double");
5.          return 1;
6.      }
7.
8.      public double aMethod(int x, double y) {
9.  B       System.out.println("int double");
10.         return 1;
11.     }
12.
13.     public int aMethod(double x, int y) {
14. C       System.out.println("double int");
15.         return 1;
16.     }
17.
18.     public double aMethod(double x, int y) {
19. D       System.out.println("double int");
20.         return 1;
21.     }
22. }
```

Ποιοι συνδυασμοί είναι αποδεκτοί;



Υπερφόρτωση (Overloading)
Προσοχή! III

- λόγω της συμβατότητας μεταξύ τύπων μια κλήση μπορεί να ταιριάζει με διάφορες μεθόδους
- καλείται αυτή που ταιριάζει ακριβώς, ή αυτή που είναι πιο κοντά
- αν υπάρχει ασάφεια θα χτυπήσει ο compiler

Υπερφόρτωση (Overloading)

Προσοχή! III - 1ο παράδειγμα


```
1.  class SomeClass {
2.     public int aMethod(int x, int y) {
3.        System.out.println("int int");
4.        return 1;
5.     }
6.
7.     public double aMethod(float x, float y) {
8.        System.out.println("float float");
9.        return 1;
10.    }
11.
12.    public int aMethod(double x, double y) {
13.       System.out.println("double double");
14.       return 1;
15.    }
16. }
17.
17. public class OverloadingExample {
18.    public static void main(String args[]) {
19.       SomeClass anObject = new SomeClass();
20.       anObject.aMethod(1,1);
21.    }
22. }
```

Τι θα τυπώσει η κλήση της μεθόδου;

Τυπώνει "int int" γιατί ταιριάζει **ακριβώς** με τις παραμέτρους που δώσαμε

Υπερφόρτωση (Overloading)

Προσοχή! III - 2ο παράδειγμα


```
1.  class SomeClass {
2.     /* public int aMethod(int x, int y) {
3.        System.out.println("int int");
4.        return 1;
5.     } */
6.
7.     public double aMethod(float x, float y) {
8.        System.out.println("float float");
9.        return 1;
10.    }
11.
12.    public int aMethod(double x, double y) {
13.       System.out.println("double double");
14.       return 1;
15.    }
16. }
17.
17. public class OverloadingExample {
18.    public static void main(String args[]) {
19.       SomeClass anObject = new SomeClass();
20.       anObject.aMethod(1,1);
21.    }
22. }
```

Τι θα τυπώσει η κλήση της μεθόδου;

Τυπώνει "float float" γιατί είναι **πιο κοντά** στις παραμέτρους που δώσαμε

Υπερφόρτωση (Overloading)

Προσοχή! III - 3ο παράδειγμα - Ασάφεια


```
1.  class SomeClass
2.  {
3.     public int aMethod(int x, double y) {
4.        System.out.println("int double");
5.        return 1;
6.     }
7.
8.     public int aMethod(double x, int y) {
9.        System.out.println("double int");
10.       return 1;
11.    }
12. }
13.
17. public class OverloadingExample
18. {
19.    public static void main(String args[]) {
20.       SomeClass anObject = new SomeClass();
21.       anObject.aMethod(1.0,1);
22.       anObject.aMethod(1,1);
23.    }
24. }
```

Τι θα τυπώσει η κλήση της μεθόδου σε κάθε περίπτωση;

Τυπώνει "double int" γιατί ταιριάζει **ακριβώς** με τις παραμέτρους που δώσαμε

ο compiler μας πετάει **λάθος** γιατί η κλήση είναι **ασαφής** (**ambiguous**)

Αντικείμενα ως ορίσματα

Αντικείμενα ως ορίσματα

- οποιαδήποτε κλάση μπορεί να χρησιμοποιηθεί ως τύπος παραμέτρου
- μπορούμε να περνάμε αντικείμενα ως ορίσματα σε μία μέθοδο όπως οποιαδήποτε άλλη μεταβλητή

Αντικείμενα ως ορίσματα

Παράδειγμα - Περιγραφή

- ορίστε μια μέθοδο που να μας επιστρέφει την απόσταση μεταξύ δύο οχημάτων

Αντικείμενα ως ορίσματα

Παράδειγμα - Υλοποίηση

```
1.  class Car {
2.     private int position = 0;
3.
4.     public Car(int position) { this.position = position; }
5.
6.     public int getPosition() { return position; }
7.
8.     public void move(int delta) { position += delta; }
9.  }
10.
13. public class MovingCarDistance1 {
14.    public static void main(String args[]) {
15.       Car myCar1 = new Car(1);
16.       Car myCar2 = new Car(0);
17.       myCar2.move(2);
18.       System.out.println("Distance of Car 1 from Car 2: " + computeDistance(myCar1, myCar2));
19.       System.out.println("Distance of Car 2 from Car 1: " + computeDistance(myCar2, myCar1));
20.    }
21.
22.    private static int computeDistance(Car car1, Car car2) {
23.       return car1.getPosition() - car2.getPosition();
24.    }
25. }
```

μια μέθοδος ή ένα πεδίο που χρησιμοποιείται σε μία **static μέθοδο** πρέπει να είναι επίσης **static**

η μέθοδος **computeDistance** παίρνει σαν όρισμα δύο **αντικείμενα** τύπου **Car**

Αντικείμενα ως ορίσματα
Παράδειγμα - Παρατήρηση

- στον αντικειμενοστραφή προγραμματισμό συνήθως δεν ορίζουμε τέτοιου είδους μεθόδους
  - η κλάση αναλαμβάνει να υλοποιεί μεθόδους που αφορούν τα αντικείμενα της

- οπότε, μέσα στην κλάση θα πρέπει να ορίσουμε μια μέθοδο που να μας δίνει την απόσταση
  - ? πώς θα το κάνουμε?
  - θα ορίσουμε μια `public` μέθοδο στην `Car`:
    - θα παίρνει σαν όρισμα ένα άλλο αντικείμενο `Car` και
    - θα μας επιστρέφει την απόσταση του από το αντικείμενο που κάλεσε τη μέθοδο

Αντικείμενα ως ορίσματα

Λεπτομέρεια

- όταν τα ορίσματα (αντικείμενα) είναι της ίδιας κλάσης με αυτή στην οποία ορίζεται η μέθοδος → τότε, η μέθοδος μπορεί να δει (έχει πρόσβαση) και στα ιδιωτικά (`private`) πεδία των αντικειμένων

- αν τα ορίσματα είναι διαφορετικού τύπου τότε η μέθοδος μπορεί να καλέσει μόνο τις `public` μεθόδους
  - (ή να δει μόνο τα `public` πεδία ← τα οποία έχουμε πει ότι τα αποφεύγουμε!)

Αντικείμενα ως ορίσματα

Διάβασμα πεδίων

- η προσπέλαση των πεδίων (για διάβασμα ή γράψιμο) γίνεται με τον ίδιο τρόπο όπως και η προσπέλαση των μεθόδων

```
<όνομα αντικειμένου>.<όνομα πεδίου>
```

Αντικείμενα ως ορίσματα

2ο παράδειγμα - Περιγραφή

- ορίστε μια μέθοδο της κλάσης `Car` που να παίρνει ως όρισμα ένα άλλο αντικείμενο `Car` και να μας επιστρέφει την απόσταση μεταξύ δύο οχημάτων

Αντικείμενα ως ορίσματα

2ο παράδειγμα - Υλοποίηση

```
1.  class Car {
2.     private int position = 0;
3.
4.     public Car(int position) { this.position = position; }
5.
6.     public void move(int delta) { position += delta; }
7.
8.     public int distanceFrom(Car other) {
9.        return this.position - other.position;
10.    }
11. }
12.
13. public class MovingCarDistance2 {
14.    public static void main(String args[]) {
15.       Car myCar1 = new Car(1);
16.       Car myCar2 = new Car(0);
17.       myCar2.move(2);
18.       System.out.println("Distance of Car 1 from Car 2: " + myCar1.distanceFrom(myCar2));
19.       System.out.println("Distance of Car 2 from Car 1: " + myCar2.distanceFrom(myCar1));
20.    }
21. }
```

Συνήθως προτιμούμε όποια μέθοδος έχει σχέση με την κλάση να την ορίζουμε ως public μέθοδο της κλάσης

Έχουμε επιπλέον ευελιξία γιατί έχουμε πρόσβαση σε όλα τα πεδία της κλάσης

Στο σημείο αυτό διαβάζουμε τα πεδία position για το αντικείμενο this και other.

Αν και το πεδίο position είναι private μπορούμε να το προσπελάσουμε γιατί είμαστε μέσα στην κλάση Car.

Μία κλάση μπορεί να προσπελάσει τα ιδιωτικά μέλη όλων των αντικειμένων της κλάσης

Αντικείμενα ως ορίσματα

Διάβασμα πεδίων II

- η προσπέλαση των πεδίων (για διάβασμα ή γράψιμο) γίνεται με τον ίδιο τρόπο όπως και η προσπέλαση των μεθόδων

```
<όνομα αντικειμένου>.<όνομα πεδίου>
```

  - και το αντικείμενο **this** είναι μια τέτοια περίπτωση

```
public int distanceFrom(Car other) {
        return this.position - other.position;
}
```

όνομα αντικειμένου | όνομα πεδίου | όνομα αντικειμένου | όνομα πεδίου

Αντικείμενα ως ορίσματα

3ο παράδειγμα - Περιγραφή

- ορίστε μια μέθοδο που θα παίρνει ως όρισμα ένα άλλο όχημα και θα βάζει το όχημα που είναι πιο πίσω στην ίδια θέση με το όχημα που είναι πιο μπροστά

Αντικείμενα ως ορίσματα

3ο παράδειγμα - Υλοποίηση

```
1.  class Car {
2.     private int position = 0;
3.
4.     public void move(int delta) { position += delta; }
5.
6.     public void catchUp(Car other) {
7.        if (this.position < other.position) {
8.           this.position = other.position;
9.        }
10.       else {
11.          other.position = this.position;
12.       }
13.    }
14.
15.    public void printPosition() {
16.       System.out.println("Car at position: " + position);
17.    }
18. }
```

μπορούμε όχι μόνο να διαβάσουμε αλλά και να **αλλάξουμε** την τιμή του πεδίου position του αντικειμένου other

```
19.  public class MovingCar16 {
20.     public static void main(String args[]) {
21.        Car myCar1 = new Car(); myCar1.move(10);
22.        Car myCar2 = new Car(); myCar2.move(20);
23.        myCar1.printPosition(); myCar2.printPosition();
24.        myCar1.catchUp(myCar2);
25.        myCar1.printPosition(); myCar2.printPosition();
26.     }
27.  }
```

Αντικείμενα ως ορίσματα

4ο παράδειγμα - Περιγραφή

υλοποιήστε μία κλάση που να χειρίζεται ένα λογαριασμό τράπεζας

- κρατάει:
  - όνομα ιδιοκτήτη και
  - ποσό

- δημιουργείστε και μία μέθοδο που συγχωνεύει δύο λογαριασμούς του ίδιου ατόμου

Αντικείμενα ως ορίσματα

4ο παράδειγμα - Υλοποίηση


```
1.  class BankAccount {
2.     private String name;
3.     private int amount;
4.
5.     public BankAccount(String name, int amount)
6.     {
7.        this.name = name;
8.        this.amount = amount;
9.     }
10.
11.    public void merge(BankAccount other) {
12.       if (this.name.equals(other.name)) {
13.          this.amount += other.amount;
14.          other.amount = 0;
15.       }
16.    }
17. }
```

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

Αντικείμενα ως ορίσματα

5ο παράδειγμα - Περιγραφή

- θέλουμε να προσομοιώσουμε την κυκλοφορία σε ένα δρόμο
  - έχουμε ένα φανάρι που μπορεί να είναι πράσινο ή κόκκινο
    - αλλάζει σε κάθε βήμα
  - έχουμε ένα όχημα που σε κάθε βήμα κινείται μία θέση, αν το φανάρι δεν είναι κόκκινο

- κλάσεις
  1. `TrafficLight`: κρατάει την κατάσταση του φαναριού και αλλάζει την κατάσταση του
  2. `Car`: τροποποίηση της `move` ώστε παίρνει όρισμα το φανάρι και να κινείται μόνο αν το φανάρι δεν είναι κόκκινο
  3. `TrafficSimulation`: κάνει την προσομοίωση

Αντικείμενα ως ορίσματα

5ο παράδειγμα - Υλοποίηση

```
1.  class TrafficLight {
2.     private boolean isLightRed = false;
3.
4.     public void change() {
5.         isLightRed = !isLightRed;
6.     }
7.
8.     public boolean isRed() {
9.         return isLightRed;
10.    }
11.
12.    public void printStatus() {
13.       if (isLightRed) {
14.          System.out.println("Traffic light
15.                              is red");
16.       }
17.       else {
18.          System.out.println("Traffic light
19.                              is green");
20.       }
21.    }
22. }
```

```
23. class Car {
24.    private int position = 0;
25.
26.    public void move(TrafficLight light) {
27.       if (!light.isRed()) { position++; }
28.    }
29.
30.    public void printPosition() {
31.       System.out.println("Car at position: " + position);
32.    }
33. }
34.
35. public class TrafficSimulation {
36.    public static void main(String args[]) {
37.       TrafficLight light = new TrafficLight();
38.       Car myCar = new Car();
39.       for (int i = 0; i < 10 ; i++) {
40.          light.printStatus();
41.          myCar.printPosition();
42.          myCar.move(light);
43.          light.change();
44.       }
45.    }
46. }
```

Αντικείμενα ως ορίσματα

5ο παράδειγμα - Υλοποίηση ΙΙ

το όρισμα στην περίπτωση αυτή είναι από **άλλη** κλάση → άρα, **δε** μπορούμε να προσπελάσουμε τα πεδία του → πρέπει να καλέσουμε τη μέθοδο **isRed()**

```
1.   class TrafficLight {
2.       private boolean isLightRed = true;
3.
4.       public void change() {
5.           isLightRed = !isLightRed;
6.       }
7.
8.       public boolean isRed() {
9.           return isLightRed;
10.      }
11.
12.      public void printStatus() {
13.          if (isLightRed) {
14.              System.out.println("Traffic light
15.                                  is red");
16.          }
17.          else {
18.              System.out.println("Traffic light
19.                                  is green");
20.          }
21.      }
22.  }
```

```
23.  class Car {
24.      private int position = 0;
25.
26.      public void move(TrafficLight light) {
27.          if (!light.isRed()) { position++; }
28.      }
29.
30.      public void printPosition() {
31.          System.out.println("Car at position: " + position);
32.      }
33.  }
34.
35.  public class TrafficSimulation {
36.      public static void main(String args[]) {
37.          TrafficLight light = new TrafficLight();
38.          Car myCar = new Car();
39.          for (int i = 0; i < 10 ; i++) {
40.              light.printStatus();
41.              myCar.printPosition();
42.              myCar.move(light);
43.              light.change();
44.          }
45.      }
46.  }
```

Αντικείμενα με πίνακες

Αντικείμενα με πίνακες

Παράδειγμα: Δυναμικός πίνακας - Περιγραφή (7ο Εργαστήριο)

Θα υλοποιήσετε ένα δυναμικό πίνακα

- ένας δυναμικός πίνακας είναι ένας πίνακας που αλλάζει χωρητικότητα ανάλογα με το πλήθος των στοιχείων που περιέχει
  - όταν ο πίνακας γεμίσει τότε η χωρητικότητα του διπλασιάζεται
  - όταν τα στοιχεία του είναι λιγότερα από το ένα τέταρτο της χωρητικότητάς του, τότε η χωρητικότητα του υποδιπλασιάζεται
- θα υλοποιήσετε την κλάση `DynamicArray`
  - κρατάει έναν πίνακα θετικών ακεραίων με
    - τα στοιχεία του πίνακα
    - τη χωρητικότητα (capacity) του πίνακα
    - το πλήθος των στοιχείων του πίνακα
- ο `constuctor` παίρνει σαν όρισμα την αρχική χωρητικότητα και αρχικοποιεί τον πίνακα

Αντικείμενα με πίνακες

Παράδειγμα: Δυναμικός πίνακας - Περιγραφή II (7ο Εργαστήριο)

η κλάση θα πρέπει να υποστηρίζει τις μεθόδους:

1. `add`: προσθέτει έναν ακέραιο στο τέλος του πίνακα
   - παίρνει το νέο ακέραιο ως όρισμα
   - αν ο πίνακας είναι γεμάτος, τότε:
     1. αντικαθιστά τον πίνακα με έναν διπλάσιας χωρητικότητας,
     2. αντιγράφει τα στοιχεία και
     3. μετά κάνει την προσθήκη του νέου στοιχείου
2. `remove`: αφαιρεί το τελευταίο στοιχείο στον πίνακα και το επιστρέφει
   - αν δεν υπάρχουν στοιχεία, επιστρέφει -1
   - αν μετά την αφαίρεση ο αριθμός των στοιχείων είναι το ένα τέταρτο της χωρητικότητας του πίνακα, τότε:
     1. αντικαθιστά τον πίνακα με έναν της μισής χωρητικότητας και
     2. αντιγράφει τα στοιχεία

Αντικείμενα με πίνακες

Παράδειγμα: Δυναμικός πίνακας - Περιγραφή III (7ο Εργαστήριο)

η κλάση θα πρέπει να υποστηρίζει τις μεθόδους:

4. πρόσβασης (accessor) για τη χωρητικότητα του πίνακα
5. `print`: τυπώνει τα στοιχεία του πίνακα

- σας δίνεται η κλάση `DynamicArrayTest` για να δοκιμάσετε τον κώδικά σας
  - αφαιρέστε τα σχόλια από τις εντολές που έχετε υλοποιήσει για να κάνετε τη δοκιμή

Αντικείμενα με πίνακες
Παράδειγμα: Δυναμικός πίνακας - Υλοποίηση

- ? τι πεδία χρειάζεται να κρατήσουμε;
  - τη χωρητικότητα (`capacity`) του πίνακα
  - τον πίνακα από ακεραίους
  - το πλήθος από ακεραίους που έχουν αποθηκευτεί (`size`)

- ? τι πρέπει να κάνει ο `constructor`;
  - να δώσει τιμή στο `capacity`
  - να δεσμεύσει χώρο στον πίνακα (`new`)

Αντικείμενα με πίνακες

Παράδειγμα: Δυναμικός πίνακας - Υλοποίηση - 1η προσπάθεια

```
1.  class DynamicArray {
2.
3.    public DynamicArray(int capacity) {
4.      int[] values = new int[capacity];
5.      int size = 0;
6.    }
7.
8.    public void print() {
9.      for (int i = 0; i < size; i++) {
10.        System.out.print(values[i] + " ");
11.     }
12.      System.out.println();
13.   }
14. }
```

? η υλοποίηση είναι **σωστή**;

ΛΑΘΟΣ!

Οι μεταβλητές capacity, size και values **δεν είναι ορισμένες**.

Για να μπορεί να τις βλέπει η μέθοδος print (ή οποιαδήποτε άλλη μέθοδος) θα **πρέπει να είναι ορισμένες** ως πεδία της κλάσης.

Αντικείμενα με πίνακες

Παράδειγμα: Δυναμικός πίνακας - Υλοποίηση - 2η προσπάθεια

```
1.  class DynamicArray {
2.    private int capacity;
3.    private int[] values;
4.    private int size = 0;
5.
6.    public DynamicArray(int capacity) {
7.      int[] values = new int[capacity];
8.    }
9.
10.   public void print() {
11.     for (int i = 0; i < size; i++) {
12.        System.out.print(values[i] + " ");
13.     }
14.      System.out.println();
15.   }
16. }
```

? η υλοποίηση είναι **σωστή**;

ΛΑΘΟΣ!

Ο constructor **δεν αρχικοποιεί** τα πεδία της κλάσης.

Οι μεταβλητές capacity και values που ορίζονται μέσα στον constructor είναι τοπικές μεταβλητές και δεν αλλάζουν την τιμή των πεδίων.

Αντικείμενα με πίνακες

Παράδειγμα: Δυναμικός πίνακας - Υλοποίηση - 3η προσπάθεια

```
1.  class DynamicArray {
2.    private int capacity;
3.    private int[] values;
4.    private int size = 0;
5.
6.    public DynamicArray(int capacity) {
7.      this.capacity = capacity;
8.    }
9.
10.   public void print() {
11.     for (int i = 0; i < size; i++) {
12.        System.out.print(values[i] + " ");
13.     }
14.      System.out.println();
15.   }
16. }
```

? η υλοποίηση είναι **σωστή**;

ΛΑΘΟΣ!

Το capacity αρχικοποιείται σωστά.

Ο πίνακας values όμως όχι.

Τον έχουμε ορίσει σωστά αλλά δεν του έχουμε δώσει χώρο! Δεν έχουμε προσδιορίσει το μέγεθος του.

Αντικείμενα με πίνακες

Παράδειγμα: Δυναμικός πίνακας - Υλοποίηση - 4η προσπάθεια

```
1.  class DynamicArray {
2.    private int capacity;
3.    private int[] values = new int[capacity];
4.    private int size = 0;
5.
6.    public DynamicArray(int capacity) {
7.      this.capacity = capacity;
8.    }
9.
10.   public void print() {
11.     for (int i = 0; i < size; i++) {
12.        System.out.print(values[i] + " ");
13.     }
14.      System.out.println();
15.   }
16. }
```

? η υλοποίηση είναι **σωστή**;

ΛΑΘΟΣ!

θυμηθείτε ότι οι εντολές αυτές θα εκτελεστούν πριν από τις εντολές του constructor!

εκείνη τη στιγμή δεν ξέρουμε το capacity → είναι μηδέν → άρα δημιουργούμε ένα πίνακα μηδενικού μεγέθους!

Αντικείμενα με πίνακες

Παράδειγμα: Δυναμικός πίνακας - Υλοποίηση - 5η προσπάθεια

```
1.  class DynamicArray {
2.    private int capacity;
3.    private int[] values;
4.    private int size = 0;
5.
6.    public DynamicArray(int capacity) {
7.      values = new int[capacity];
8.    }
9.
10.   public void add() {
11.     if (size == capacity) {
12.         ...
13.     }
14. }
```

? η υλοποίηση είναι **σωστή**;

ΛΑΘΟΣ!

Ο Constructor θα αρχικοποιήσει σωστά τον πίνακα values, αλλά δεν θα αλλάξει το πεδίο capacity μιας και χρησιμοποιεί την τοπική μεταβλητή

Το capacity εδώ αναφέρεται στο πεδίο και έχει τιμή μηδέν, άρα ο έλεγχος θα βγαίνει αληθής

Αντικείμενα με πίνακες

Παράδειγμα: Δυναμικός πίνακας - Υλοποίηση - 6η Προσπάθεια

```
1.  class DynamicArray {
2.    private int capacity;
3.    private int[] values;
4.    private int size = 0;
5.
6.    public DynamicArray(int capacity) {
7.      this.capacity = capacity;
8.      values = new int[capacity];
9.    }
10.
11.   ...
12. }
```

? η 6η υλοποίηση είναι **σωστή**;

1. πρώτα δηλώνουμε τα **πεδία** μέσα στην κλάση

2. μετά δίνουμε τιμή στη διάσταση και αφού πλέον ξέρουμε τη διάσταση δίνουμε χώρο στον πίνακα που θα κρατάει τις τιμές

3. τώρα μπορούμε να **κάνουμε** και την **αρχικοποίηση**

Αντικείμενα με πίνακες

Παράδειγμα: Δυναμικός πίνακας - Υλοποίηση - Η μέθοδος add

η μέθοδος **`add`**

1. προσθέτει ένα ακέραιο στον πίνακα και
2. αν γεμίσει ο πίνακας διπλασιάζει το μέγεθοστου πίνακα

❓ τι σημαίνει αν γεμίσει ο πίνακας?

- το `size` (αριθμός στοιχείων) είναι ίδιο με το `capacity` (χωρητικότητα/μέγεθος) του πίνακα

❓ τι σημαίνει διπλασιάζει το μέγεθος του πίνακα?

✍ οι πίνακες έχουν σταθερό μέγεθος

i. πρέπει να δημιουργήσουμε ένα καινούριο πίνακα με διπλάσια χωρητικότητα
ii. να αντιγράψουμε τα στοιχεία που έχουμε σε αυτό τον πίνακα
iii. να κάνουμε το πεδίο `values` της κλάσης να δείχνει στο νέο πίνακα
iv. να ενημερώσουμε το πεδίο `capacity`

Αντικείμενα με πίνακες

Παράδειγμα: Δυναμικός πίνακας - Υλοποίηση - Η μέθοδος add

```
1.  class DynamicArray {
2.    private int capacity;
3.    private int[] values;
4.    private int size = 0;
5.
6.    public void add(int x) {
7.      if(size == capacity) {
8.          capacity = 2*capacity;
9.          int[] temp = new int[capacity];
10.         for (int i = 0; i < size; i++) {
11.             temp[i] = values[i];
12.         }
13.         values = temp;
14.     }
15.     values[size] = x;
16.     size++;
17.   }
18.   ...
19. }
```

ενημερώνουμε το πεδίο capacity

ενημερώνουμε το πεδίο values

ενημερώνουμε το πεδίο size

αντιγράφουμε τα στοιχεία

προσθέτουμε το νέο στοιχείο στον πίνακα

το πεδίο size μας δίνει τόσο το πλήθος των στοιχείων όσο και την επόμενη κενή θέση, στον πίνακα

Αντικείμενα με πίνακες

Παράδειγμα: Δυναμικός πίνακας - Υλοποίηση - Δημιουργία νέου πίνακα

- ? τι ακριβώς γίνεται όταν δημιουργούμε ένα νέο πίνακα;

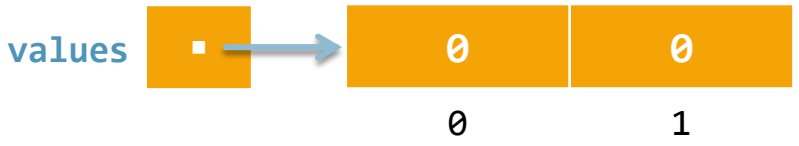
- η εντολή στο δημιουργό:

```
values = new int[capacity];
```

δημιουργεί χώρο στη μνήμη και βάζει το πεδίο **values** να δείχνει σε αυτόν

π.χ.

- έστω ότι το **capacity** έχει τιμή 2:



- μετά από την εκτέλεση **add(2)** και **add(9)**:

Αντικείμενα με πίνακες

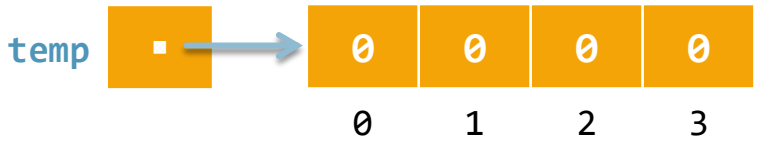
Παράδειγμα: Δυναμικός πίνακας - Υλοποίηση - Δημιουργία νέου πίνακα III

- ? τι ακριβώς γίνεται όταν δημιουργούμε ένα νέο πίνακα;

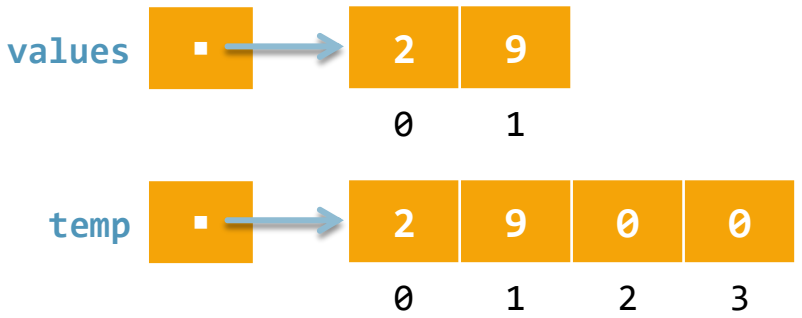
- η εντολή στην **add**:

```
int[] temp = new int[2*capacity];
```

δημιουργεί χώρο στη μνήμη και βάζει την τοπική μεταβλητή **temp** να δείχνει σε αυτόν



- με το βρόχο **for** αντιγράφουμε τα στοιχεία από το **values** στο **temp**



Αντικείμενα με πίνακες

Παράδειγμα: Δυναμικός πίνακας - Υλοποίηση - Δημιουργία νέου πίνακα II

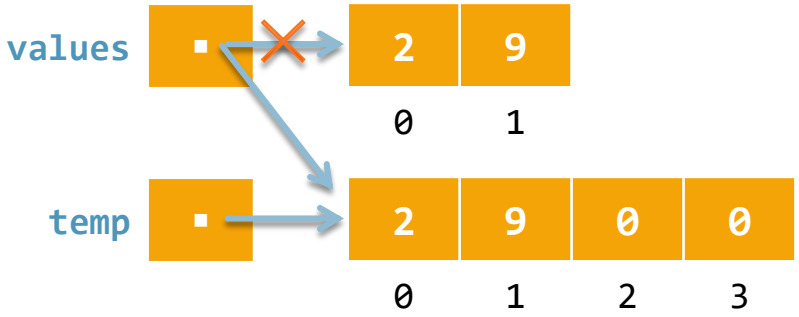
- ? τι ακριβώς γίνεται όταν δημιουργούμε ένα νέο πίνακα;

- με την εντολή:

```
values = temp;
```

βάζουμε το πεδίο `values` να δείχνει στο χώρο που δείχνει η μεταβλητή `temp`

  - ο προηγούμενος χώρος χάνεται
  - το `values` είναι πεδίο και άρα ο νέος χώρος διατηρείται και μετά το τέλος της `add`, αφού εξαφανιστεί η `temp`



Αντικείμενα με πίνακες

Παράδειγμα: Δυναμικός πίνακας - Υλοποίηση - Η μέθοδος remove

η μέθοδος **`remove`**

1. αφαιρεί το τελευταίο στοιχείο και το επιστρέφει
2. αν το πλήθος των στοιχείων πέσει στο ¼ →υποδιπλασιάζει το μέγεθος του πίνακα

? τι σημαίνει αφαιρεί?

- το πλήθος των στοιχείων (**`size`**) μας λέει πόσα στοιχεία έχει ο πίνακας
- αν μειώσουμε το **`size`** κατά 1, ουσιαστικά αφαιρούμε το τελευταίο στοιχείο
- δε χρειάζεται να το «σβήσουμε»

Αντικείμενα με πίνακες

Παράδειγμα: Δυναμικός πίνακας - Υλοποίηση - Η μέθοδος remove

```
1.  class DynamicArray {
2.    public int remove() {
3.      if(size == 0) {
4.        return -1;
5.      }
6.      size --;
7.      int retValue = values[size];
8.      if(size <= capacity/4) {
9.        capacity = capacity/2;
10.       int[] temp = new int[capacity];
11.       for (int i = 0; i < size; i++) {
12.           temp[i] = values[i];
13.       }
14.       values = temp;
15.     }
16.     return retValue;
17.   }
18.   ...
19. }
```

ελέγχουμε την περίπτωση του άδειου πίνακα

ενημερώνουμε το πεδίο size

κρατάμε την τιμή του στοιχείου που θέλουμε να πιστρέψουμε (το οποίο είναι στη θέση size-1)

ενημερώνουμε το πεδίο capacity

δημιουργούμε πίνακα μισού μεγέθους και αντιγράφουμε τα στοιχεία

ενημερώνουμε το πεδίο values

επιτρέφουμε την τιμή

Αντικείμενα με πίνακες

Παράδειγμα: Δυναμικός πίνακας - Υλοποίηση - Μέθοδος ανάγνωσης (accessor) για το capacity

- οι μέθοδοι ανάγνωσης (accessors), όπως και οι μέθοδοι τροποποίησης (mutators), έχουν πολύ συγκεκριμένο όνομα και συγκεκριμένη λειτουργία

- όνομα για ανάγνωση:

  `get<όνομα πεδίου>()`

  - στην περίπτωσή μας: `getCapacity()`

- λειτουργία:
  - επιστρέφουν την τιμή του πεδίου
  - δεν εκτυπώνουν!

```
1.  class DynamicArray {
2.    private int capacity;
3.
4.    public int getCapacity() {
5.      return capacity;
6.    }
7.    ...
8.  }
```

Αντικείμενα με πίνακες

Παράδειγμα: Δυναμικός πίνακας - Υλοποίηση - Η κλάση DynamicArrayTest

```
1.  class DynamicArrayTest {
2.     public static void main(String[] args) {
3.        // DynamiArray array = new DynamicArray(2);
4.        // System.out.println("Removed: "+ array.remove());
5.        // array.add(2);
6.        // array.add(4);
7.        // array.add(9);
8.        // Print capacity of array
9.        // System.out.println("The array has " + array.getCapacity() + " elements");
10.       // Print elements of array
11.       // array.print();
12.       // System.out.println("Removed: "+ array.remove());
13.       // System.out.println("Removed: "+ array.remove());
14.       // Print capacity of array
15.       // System.out.println("The array has " + array.getCapacity() + " elements");
16.       // Print elements of array
17.       // array.print();
18.    }
19. }
```

1. τα προγράμματα σας πρέπει να τα δημιουργείτε **κομμάτι-κομμάτι**
2. κάθε φορά που **ολοκληρώνετε** μια μέθοδο πρέπει να τη **δοκιμάζετε**
3. αυτός είναι ο στόχος της **DynamicArrayTest**

Οι μέθοδοι toString και equals

Δύο ειδικές μέθοδοι

η Java «περιμένει» να δει τις εξής δύο μεθόδους για κάθε αντικείμενο

1. `toString`: επιστρέφει μία `String` αναπαράσταση του αντικειμένου
2. `equals`: ελέγχει για ισότητα δύο αντικειμένων

και οι δύο συναρτήσεις ορίζονται από τον προγραμματιστή

- μπορούμε να ορίσουμε όπως μας βολεύει:
  - τι `String` θα επιστραφεί
  - τι σημαίνει δύο αντικείμενα να είναι ίσα

Δύο ειδικές μέθοδοι
toString και equals

- η μέθοδος `toString` ορίζεται πάντα ως:

```
public String toString() {
    ...
}
```

  - αν έχουμε ορίσει την `toString` μπορούμε να χρησιμοποιήσουμε τα αντικείμενα της κλάσης σαν `String`
    - καλείται αυτόματα η `toString`

- η μέθοδος `equals` ορίζεται πάντα ως:

```
public boolean equals(<class name> other) {
    ...
}
```

Δύο ειδικές μέθοδοι

toString και equals - 1ο παράδειγμα - Περιγραφή

- στην κλάση `Car` θέλουμε να προσθέσουμε τις μεθόδους `toString` και `equals`
  - η `toString` θα επιστρέφει ένα `String` με τη θέση του αυτοκινήτου
  - η `equals` θα ελέγχει αν δύο οχήματα έχουν την ίδια θέση

Δύο ειδικές μέθοδοι

toString - 1ο παράδειγμα - 1η υλοποίηση

```
1.  class Car {
2.     private Integer position;
3.
4.     public Car(Integer position) { this.position = position; }
5.
6.     public void move(int delta) {position += delta; }
7.
13.    public String toString() {
14.       return position.toString();
15.    }
16. }
17.
18. public class MovingCarToString {
19.    public static void main(String args[]) {
20.       Car myCar1 = new Car(1);
21.       Car myCar2 = new Car(0);
22.       System.out.println("car 1 is at " + myCar1 + "and car 2 is at " + myCar2);
23.    }
24. }
```

για να μπορούμε να μετατρέψουμε τον ακέραιο σε String ορίζουμε το position ως Integer (wrapper class)

Η Java περιμένει αυτό το συντακτικό για τον ορισμό της toString

καλούμε τη συνάρτηση toString() της κλάσης Integer

χρησιμοποιούμε τα αντικείμενα myCar1, myCar2 ως String → καλείται η μέθοδος toString() αυτόματα

ισοδύναμο με το:

```
System.out.println("car 1 is at " + myCar1.toString() + "and car 2 is at " + myCar2.toString());
```

Δύο ειδικές μέθοδοι

toString - 1ο παράδειγμα - 2η υλοποίηση

```
1.  class Car {
2.     private int position;
3.
4.     public Car(int position) { this.position = position; }
5.
6.     public void move(int delta) {position += delta; }
7.
13.    public String toString() {
14.       return "" + position;
15.    }
16. }
17.
18. public class MovingCarToString {
19.    public static void main(String args[]) {
20.       Car myCar1 = new Car(1);
21.       Car myCar2 = new Car(0);
22.       System.out.println("car 1 is at " + myCar1 + "and car 2 is at " + myCar2);
23.    }
24. }
```

Ένας άλλος τρόπος να μετατρέψουμε ένα int σε String

Δύο ειδικές μέθοδοι

equals - 1ο παράδειγμα - Υλοποίηση

```
1.  class Car {
2.     private int position = 0;
3.
4.     public Car(int position) { this.position = position; }
5.
6.     public void move(int delta) {position += delta; }
7.
13.    public boolean equals(Car other) {
14.       if (this.position == other.position) {
15.             return true;
16.       }
17.       else { return false; }
18.    }
19. }
20.
21. public class MovingCarToString {
22.    public static void main(String args[]) {
23.       Car myCar1 = new Car(2);
24.       Car myCar2 = new Car(0); myCar2.move(2);
25.       if (myCar1.equals(myCar2)) {
26.          System.out.println("collision!");
27.    }
28. }
```

Η Java περιμένει αυτό το συντακτικό για τον ορισμό της equals

Στο σημείο αυτό διαβάζουμε τα πεδία position για το αντικείμενο this και other.

Αν και το πεδίο position είναι private μπορούμε να το προσπελάσουμε γιατί είμαστε μέσα στην κλάση Car.

Μία κλάση μπορεί να προσπελάσει τα ιδιωτικά μέλη όλων των αντικειμένων της κλάσης

κλήση της equals() στο πρόγραμμα

Δύο ειδικές μέθοδοι

toString και equals - 2ο παράδειγμα - Περιγραφή

- ορισμός μεθόδων `toString` και `equals` στην κλάση `Person`

Δύο ειδικές μέθοδοι

toString και equals - 2ο παράδειγμα - 1η υλοποίηση

```
1.  class Person {
2.     private String name;
3.
4.     public Person(String name) { this.name = name; }
5.
6.     public String toString() { return name; }
7.
8.     public boolean equals(Person other) {
9.        return this.name.equals(other.name);
10.    }
11. }
12.
13. public class TwoPersons {
14.    public static void main(String args[]) {
15.       Person alice = new Person("Alice");
16.       Person bob = new Person("Bob");
17.       if (!alice.equals(bob)) {
18.          System.out.println("There are two different persons: " + alice + "and " + bob);
19.    }
20. }
```

Δύο ειδικές μέθοδοι

toString και equals - 2ο παράδειγμα - 2η υλοποίηση

```
1.  class Person {
2.     private String firstName;
3.     private String lastName;
4.
5.     public Person(String firstName, String lastName) {
6.        this.firstName = firstName; this.lastName = lastName;
7.     }
8.
9.     public String toString() { return firstName + " " + lastName; }
10.
11.    public boolean equals(Person other) {
12.       return (this.firstName.equals(other.firstName) && this.lastName.equals(other.lastName));
13.    }
14. }
15.
16. public class TwoPersons2 {
17.    public static void main(String args[]) {
18.       Person alice = new Person("Alice", "Wonderland");
19.       Person bob = new Person("Bob", "Sfougkarakis");
20.       if (!alice.equals(bob)) {
21.          System.out.println("There are two different persons: " + alice + "and " + bob);
22.       }
23.    }
24. }
```

Αντικείμενα μέσα σε αντικείμενα

Αντικείμενα μέσα σε αντικείμενα

- τα αντικείμενα εκτός από ορίσματα σε μεθόδους οποιασδήποτε κλάσης μπορούν να εμφανιστούν και ως πεδία μιας κλάσης
  - ένα αντικείμενο μπορεί να έχει μέσα του άλλα αντικείμενα

Αντικείμενα μέσα σε αντικείμενα

1ο Παράδειγμα - Περιγραφή

- θέλουμε να επεκτείνουμε την κλάση `Car` ώστε να γνωρίζουμε τον οδηγό (`Person`) κάθε οχήματος

Αντικείμενα μέσα σε αντικείμενα

1ο παράδειγμα - 1η υλοποίηση

```
1.  class Person {
2.     private String name;
3.
4.     public Person(String name) {
5.        this.name = name;
6.     }
7.
8.     public String getName() {
9.        return name;
10.    }
11. }
12.
13. class Car {
14.    private int position = 0;
15.    private Person driver;
16.
17.    public Car(int position, String name) {
18.       this.position = position;
19.       this.driver = new Person(name);
20.    }
21.
22.    public String toString() { return driver.getName() + " " + position; }
23. }
```

```
24. public class MovingCarDriver1 {
25.    public static void main(String args[]) {
26.       Car myCar = new Car(1, "Alice");
27.       System.out.println(myCar);
28.    }
29. }
```

το αντικείμενο δημιουργείται μέσα στον constructor

αυτό έχει νόημα αν το Person χρησιμοποιείται **μόνο** μέσα στην κλάση Car

Αντικείμενα μέσα σε αντικείμενα

1ο παράδειγμα - 2η υλοποίηση

```
1.  class Person {
2.     private String name;
3.
4.     public Person(String name) {
5.        this.name = name;
6.     }
7.
8.     public String getName() {
9.        return name;
10.    }
11. }
12.
13. class Car {
14.    private int position = 0;
15.    private Person driver;
16.
17.    public Car(int position, Person driver) {
18.       this.position = position;
19.       this.driver = driver;
20.    }
21.
22.    public String toString() { return driver.getName() + " " + position; }
23. }
```

```
24. public class MovingCarDriver2 {
25.    public static void main(String args[]) {
26.       Person alice = new Person("Alice");
27.       Car myCar = new Car(1, alice);
28.       System.out.println(myCar);
29.    }
30. }
```

καλύτερη υλοποίηση!

Αντικείμενα μέσα σε αντικείμενα

2ο Παράδειγμα - Περιγραφή

- θέλουμε να επεκτείνουμε την κλάση `Car` ώστε να γνωρίζουμε τον οδηγό (`Person`) κάθε οχήματος
  - ένας οδηγός πρέπει απαραίτητα να είναι άνω των 18 ετών

Αντικείμενα μέσα σε αντικείμενα

1ο παράδειγμα - 1η υλοποίηση

```
1.  class Person {
2.     private String name;
3.     private int age;
4.
5.     public Person(String name, int age) {
6.        this.name = name;
7.        this.age = age;
8.     }
9.
10.    public String getName() {
11.       return name;
12.    }
13.
14.    public int getAge() {
15.       return age;
16.    }
17. }
```

τι κάνουμε εάν ο οδηγός είναι μικρότερος από 18 χρονών;

```
18. class Car {
19.    private int position;
20.    private Person driver;
21.
22.    public Car(int position, Person driver) {
23.       this.position = position;
24.       if (driver.getAge() >= 18) {
25.          this.driver = driver;
26.       }
27.    }
28.
29.    public String toString() {
30.       return driver.getName() + " " + position;
31.    }
32. }
33.
24. public class MovingCarDriver3 {
25.    public static void main(String args[]) {
26.       Person alice = new Person("Alice", 20);
27.       Car myCar = new Car(1, alice);
28.       System.out.println(myCar);
29.    }
30. }
```

Η εντολή exit

```
1.  class Car {
2.    private int position;
3.    private Person driver;
4.
5.    public Car(int position, Person driver) {
6.      this.position = position;
7.      if (driver.getAge() >= 18) {
8.         this.driver = driver;
9.      }
10.     else {
11.        System.exit(-1);
12.     }
13.   }
14.
15.   public String toString() {
16.     return driver.getName() + " " + position;
17.   }
18. }
```

- χρησιμοποιείται για σοβαρά λάθη για να σταματάει την εκτέλεση του προγράμματος

αν δώσουμε **μη αποδεκτή** ηλικία το πρόγραμμα μας θα **σταματήσει**

το -1 εξυπηρετεί σαν **κωδικός λάθους** (μπορείτε να βάλετε **όποια** τιμή θέλετε)

Αντικείμενα μέσα σε αντικείμενα

toString και equals - Παράδειγμα - Περιγραφή

```
1.  class Person {
2.     private String name;
3.     private int age;
4.
5.     public Person(String name, int age) {
6.        this.name = name;
7.        this.age = age;
8.     }
9.  }
```

```
10.  class Car {
11.     private int position;
12.     private Person driver;
13.
14.     public Car(int position, Person driver) {
15.        this.position = position;
16.         this.driver = driver;
17.     }
18.  }
```

? πώς θα υλοποιήσουμε την `toString` και την `equals`;

Αντικείμενα μέσα σε αντικείμενα

toString και equals - Παράδειγμα - Υλοποίηση

```
1.  class Person {
2.     private String name;
3.     private int age;
4.
5.     public Person(String name, int age) {
6.        this.name = name;
7.        this.age = age;
8.     }
9.
10.    public String toString() {
11.        return name + " " + age;
12.    }
13.
13.    public boolean equals(Person other) {
14.       return (
15.              this.name.equals(other.name)
                 &&
                 this.age.equals(other.age)
16.       );
17.    }
18. }
```

```
19. class Car {
20.    private int position;
21.    private Person driver;
22.
23.    public Car(int position, String name) {
24.       this.position = position;
25.       this.driver = driver;
26.    }
27.
28.    public String toString() {
29.       return driver + " " + position;
30.    }
31.
32.    public boolean equals(Car other) {
33.       return (this.position == other.position &&
                  this.driver.equals(other.driver));
34.    }
35. }
```

φωλιασμένη κλήση της **toString** και της **equals**

Δύο ειδικές μέθοδοι

toString και equals - Φωλιασμένες κλήσεις

- ένα αντικείμενο μπορεί να περιέχει μέσα άλλα αντικείμενα
- τότε, είναι συνηθισμένο οι μέθοδοι `toString` και `equals` να ορίζονται κάνοντας φωλιασμένη κλήση της `toString` και της `equals` επί των αντικειμένων που περιέχει
  - φωλιασμένη κλήση της `toString` της κλάσης `Person`:

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

  - φωλιασμένη κλήση της `equals` της κλάσης `Person`:

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

Αντικείμενα μέσα σε αντικείμενα

2ο παράδειγμα: Τραπεζικές συναλλαγές - Περιγραφή (2ο Πρόγραμμα - 6ο Εργαστήριο)

δημιουργήστε την κλάση **`BankAccountPlus`** η οποία

- κρατάει πληροφορία για έναν τραπεζικό λογαριασμό
  - έχει ένα πεδίο: έναν πραγματικό αριθμό για το ποσό που έχει ο λογαριασμός
  - έχει τρεις μεθόδους:
    1. `deposit`: προσθέτει χρήματα
    2. `withdraw`: αφαιρεί χρήματα
    3. `printStatement`: εκτυπώνει την κατάσταση του λογαριασμού
- κρατάει και ένα `ArrayList` από `String` στο οποίο διατηρεί τη λίστα με τις συναλλαγές του λογαριασμού
  - για κάθε συναλλαγή θα προσθέτει ένα `String` που θα περιγράφει τη συναλλαγή και το ποσό της συναλλαγής
  - η μέθοδος `printStatement` εκτυπώνει και τη λίστα των συναλλαγών

Αντικείμενα ως επιστρεφόμενες τιμές

Αντικείμενα ως επιστρεφόμενες τιμές

- μια μέθοδος μπορεί να επιστρέφει αντικείμενα όπως οποιαδήποτε άλλη τιμή

- είναι δυνατόν επίσης μέσα σε μία μέθοδο να δημιουργούμε ένα αντικείμενο και να το επιστρέφουμε για να χρησιμοποιηθεί

Αντικείμενα ως επιστρεφόμενες τιμές

1ο παράδειγμα

```
1.  class Car {
2.     private int position;
3.     private Person driver;
4.
5.     public Car(int position, String name) {
6.        this.position = position;
7.        this.driver = new Person(name);
8.     }
9.
10.    public String toString() {
11.       return driver + " " + position;
12.    }
13.
14.    public Person getDriver() {
15.       return driver;
16.    }
17. }
```

επιστρέφει το αντικείμενο Person το οποίο είναι ο οδηγός του οχήματος

Αντικείμενα ως επιστρεφόμενες τιμές

2ο παράδειγμα - Περιγραφή

υλοποιήστε μία κλάση που να χειρίζεται ένα λογαριασμό τράπεζας

- κρατάει:
  - όνομα ιδιοκτήτη και
  - ποσό

- δημιουργείστε και μία μέθοδο που συγχωνεύει δύο λογαριασμούς του ίδιου ατόμου, δημιουργώντας έναν καινούργιο λογαριασμό και επιστρέφοντάς τον

Αντικείμενα ως επιστρεφόμενες τιμές

2ο παράδειγμα


```
1.  class BankAccount {
2.     private String name;
3.     private int amount;
4.
5.     public BankAccount(String name, int amount)
6.     {
7.        this.name = name;
8.        this.amount = amount;
13.    }
14.
15.    public BankAccount merge(BankAccount other) {
16.       if (this.name.equals(other.name)) {
17.          BankAccount newAccount = new BankAccount(name, this.amount + other.amount);
18.          this.amount = 0;
19.          other.amount = 0;
20.          return newAccount;
21.       }
22.       return null;
23.    }
24. }
```

δημιουργούμε ένα νέο αντικείμενο BankAccount και το επιστρέφουμε

αν **δε μπορούμε** να δημιουργήσουμε το νέο λογαριασμό επιστρέφουμε **null** (το null είναι το **κενό αντικείμενο**)

Σύνοψη

- Δημιουργός (Constructor)
  - συντακτικό
  - παραδείγματα
- Υπερφόρτωση (Overloading)
  - μεθόδων
  - δημιουργών
  - παραδείγματα
  - ασάφεια
- Αντικείμενα
  - ως ορίσματα
  - σε πίνακες
  - μέσα σε αντικείμενα
  - ως επιστρεφόμενες τιμές
- Ειδικές μέθοδοι `toString` και `equals` της Java