ΤΕΧΝΙΚΕΣ ΑΝΤΙΚΕΙΜΕΝΟΣΤΡΑΦΟΥΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ

Αντικείμενα με πίνακες

Μέθοδοι toString και equals

Αντικείμενα μέσα σε αντικείμενα

ΑΝΤΙΚΕΙΜΕΝΑ ΜΕ ΠΙΝΑΚΕΣ

Θα υλοποιήσετε την κλάση **DynamicArray**. Η κλάση θα κρατάει ένα πίνακα θετικών ακεραίων με τα στοιχεία του πίνακα, την χωρητικότητα (**capacity**) του πίνακα, και τον αριθμό των στοιχείων. Ο constuctor παίρνει σαν όρισμα την αρχική χωρητικότητα και αρχικοποιεί τον πίνακα. Η κλάση θα πρέπει να έχει επίσης τις μεθόδους:

- **add**: Προσθέτει ένα νέο ακέραιο στο τέλος του πίνακα (υποθέστε ότι ο αριθμός είναι θετικός, δεν χρειάζεται να κάνετε τον έλεγχο). Αν ο πίνακας είναι γεμάτος, τότε αντικαθιστά τον πίνακα με έναν διπλάσιας χωρητικότητας, αντιγράφει τα στοιχεία και μετά κάνει την προσθήκη του νέου στοιχείου
- **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 αντικαθιστά τον πίνακα με έναν της μισής χωρητικότητας και αντιγράφει τα στοιχεία.
- Την μέθοδο **πρόσβασης** (**accessor**) για τη χωρητικότητα του πίνακα.
- **print**: Τυπώνει τα στοιχεία του πίνακα.

Πεδία

- Τι πεδία χρειάζεται να κρατήσουμε?
  - Την χωρητικότητα capacity του πίνακα
  - Τον πίνακα από ακεραίους
  - Τον αριθμό από ακεραίους που έχουν αποθηκευτεί (size)

- Τι πρέπει να κάνει ο constructor?
  - Να δώσει τιμή στο capacity
  - Να δώσει χώρο στον πίνακα (new)

```
class DynamicArray
{
   public DynamicArray(int capacity)
   {
        int[] values = new int[capacity];
        int size = 0;
   }

   public void print()
   {
        for (int i = 0; i < size; i ++){
          System.out.println(values[i] + " “);
        }
        System.out.println();
   }
}
```

Σωστό ή λάθος?

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

ΛΑΘΟΣ!

```
class DynamicArray
{
  private int capacity;
  private int[] values;
  private int size = 0;

  public DynamicArray(int capacity)
  {
      int[] values = new int[capacity];
  }

  public void print()
  {
      for (int i = 0; i < size; i ++){
        System.out.println(values[i] + " ");
      }
      System.out.println();
  }
}
```

Σωστό?

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.

ΛΑΘΟΣ!

```
class DynamicArray
{
  private int capacity;
  private int[] values;
  private int size = 0;

  public DynamicArray(int capacity)
  {
     this.capacity = capacity;
  }

  public void print()
  {
       for (int i = 0; i < size; i ++){
         System.out.println(values[i] + " “);
       }
       System.out.println();
  }
}
```

Σωστό?

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

ΛΑΘΟΣ!

```
class DynamicArray
{
  private int capacity;
  private int[] values = new int[capacity];
  private int size = 0;

  public DynamicArray(int capacity)
  {
     this.capacity = capacity;
  }

  public void print()
  {
       for (int i = 0; i < size; i ++){
         System.out.println(values[i] + " ");
       }
       System.out.println();
  }
}
```

Σωστό?

Θυμηθείτε ότι οι εντολές αυτές θα εκτελεστούν πριν από τις εντολές του constructor. Εκείνη τη στιγμή δεν ξέρουμε το capacity είναι μηδέν και άρα δημιουργούμε ένα πίνακα μηδενικού μεγέθους!

ΛΑΘΟΣ!

```
class DynamicArray
{
   private int capacity;
   private int values[];
   private int size = 0;

   public DynamicArray(int capacity)
   {
        values = new int[capacity];
   }

   public void add(int x)
   {
        if (size == capacity){
                .....
        }
   }
}
```

Σωστό?

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

ΛΑΘΟΣ!

```
class DynamicArray
{
  private int capacity;
  private int values[];
  private int size = 0;

  public DynamicArray(int capacity)
  {
      this.capacity = capacity;
      values = new int[capacity];
  }

}
```

Σωστό?

Πρώτα δηλώνουμε τα πεδία μέσα στην κλάση

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

Τώρα μπορούμε και να κάνουμε και την αρχικοποίηση

ΣΩΣΤΟ!

Η μέθοδος add

- Η μέθοδος add προσθέτει ένα ακέραιο στον πίνακα και αν γεμίσει ο πίνακας διπλασιάζει το μέγεθος του 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
  - Πρέπει να δημιουργήσουμε ένα καινούριο πίνακα με διπλάσια χωρητικότητα
  - Να αντιγράψουμε τα στοιχεία που έχουμε σε αυτό τον πίνακα
  - Να κάνουμε το πεδίο values της κλάσης να δείχνει στον νέο πίνακα.
  - Να ενημερώσουμε το πεδίο capacity

Η μέθοδος add

```
public void add(int x)
{
    if (size == capacity){
        capacity = 2*capacity;
        int[] temp = new int[capacity];
        for (int i = 0; i < size; i ++){
            temp[i] = values[i];
        }
        values = temp;
    }
    values[size] = x;
    size ++;
}
```

Ενημερώνει το πεδίο capacity

Αντιγράφει τα στοιχεία

Ενημερώνει το πεδίο values

Ενημερώνει το πεδίο size

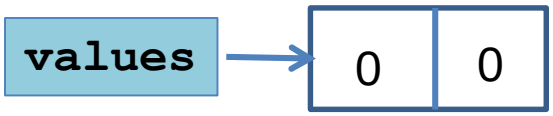
Προσθέτει το νέο στοιχείο στον πίνακα. Το πεδίο size μας δίνει και τον αριθμό των στοιχείων στον πίνακα αλλά και την επόμενη αδειανή θέση.

Δημιουργία νέου πίνακα

- Τι ακριβώς γίνεται όταν δημιουργούμε το νέο πίνακα?
- Η εντολή στον constructor

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

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



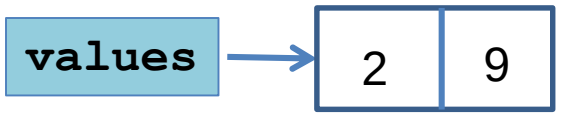
Δημιουργία νέου πίνακα

- Τι ακριβώς γίνεται όταν δημιουργούμε το νέο πίνακα?
- Η εντολή στον constructor

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

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

- Μετά από δύο add

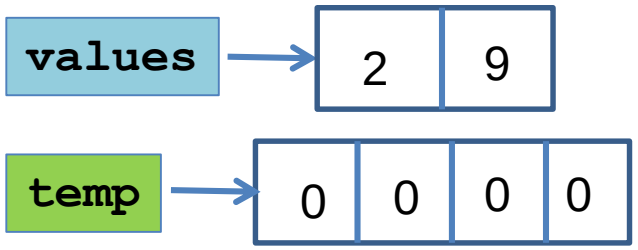


Δημιουργία νέου πίνακα

- Η εντολή στην add

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

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



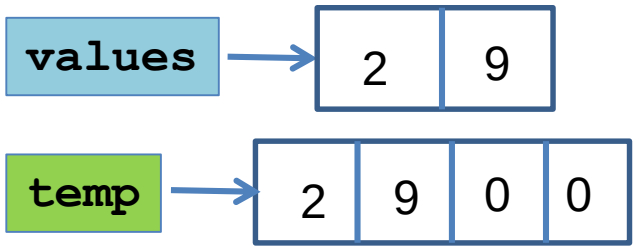
Δημιουργία νέου πίνακα

- Η εντολή στην add

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

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

- Με το for loop αντιγράφουμε τα στοιχεία από το values στο temp



Δημιουργία νέου πίνακα

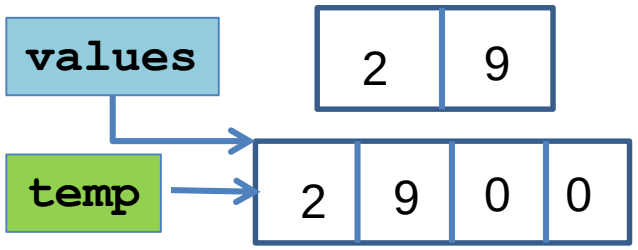
- Η εντολή στην add

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

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

- Με το for loop αντιγράφουμε τα στοιχεία από το values στο temp
- Με την εντολή

```
values = temp
```



Βάζουμε το πεδίο values να δείχνει στον χώρο που δείχνει η μεταβλητή temp

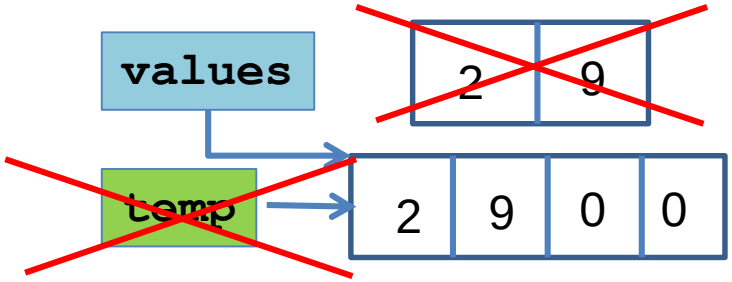
Δημιουργία νέου πίνακα

- Με την εντολή

```
values = temp
```

Βάζουμε το πεδίο values να δείχνει στον 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.



Η μέθοδος remove

- Η remove αφαιρεί το τελευταίο στοιχείο και το επιστρέφει. Αν ο αριθμός των στοιχείων πέσει στο ¼ υποδιπλασιάζει το μέγεθος του πίνακα
- Τι σημαίνει αφαιρεί?
  - Ο αριθμός των στοιχείων μας λέει πόσα στοιχεία «βλέπουμε». Αν μειώσουμε τον αριθμό κατά 1, ουσιαστικά αφαιρούμε το τελευταίο στοιχείο. Δεν χρειάζεται να το «σβήσουμε».

Η μέθοδος remove

```
public int remove()
{
        if (size == 0){
                return -1;
        }
        size -- ;
        int retValue = values[size];
        if (size == capacity/4){
                capacity = capacity/2;
                int[] temp = new int[capacity];
                for (int i = 0; i < size; i ++){
                        temp[i] = array[i];
                }
                values = temp;
        }
        return retValue;
}
```

Ελέγχει την περίπτωση του άδειου πίνακα

Κρατάει το στοιχείο που θέλουμε να επιστρέψουμε. Είναι στην θέση size-1

Ενημερώνει το πεδίο size

Ενημερώνει το πεδίο capacity

Δημιουργεί πίνακα μισού μεγέθους και αντιγράφει τα στοιχεία

Ενημερώνει το πεδίο values

Επιστρέφει την τιμή

Accessor για το capacity

- Οι accessor μέθοδοι (όπως και οι mutator) έχουν ένα πολύ συγκεκριμένο όνομα, και συγκεκριμένη λειτουργία
- Όνομα:
  - get<Ονομα Πεδιου>()
  - Στην περίπτωση μας getCapacity()
- Λειτουργία:
  - Επιστρέφουν την τιμή του πεδίου. Δεν εκτυπώνουν!

```
public int getCapacity()
{
        return capacity;
}
```

Η κλάση DynamicArrayTest

```
class DynamicArrayTest
{
   public static void main(String[] args){
       //DynamicArray array = new DynamicArray(2);
       //System.out.println("Removed:"+array.remove());
       //array.add(2);
       //array.add(4);
       //array.add(9);
       // Εκτυπώστε το capcity του array
       //array.print();
       //System.out.println("Removed:"+array.remove());
       //System.out.println("Removed:"+array.remove());
       // Εκτυπώστε το capcity του array
       //array.print();
   }
}
```

Τα προγράμματα σας πρέπει να τα δημιουργείτε κομμάτι-κομμάτι. Κάθε φορά που ολοκληρώνετε μια μέθοδο πρέπει να την τεστάρετε. Αυτός ήταν ο στόχος της DynamicArrayTest

ΟΙ ΜΕΘΟΔΟΙ TOSTRING ΚΑΙ EQUALS

Δυο ειδικές μέθοδοι

- Η Java «περιμένει» να δει τις εξής δύο μεθόδους για κάθε αντικείμενο
  - Τη μέθοδος toString η οποία για ένα αντικείμενο επιστρέφει μία string αναπαράσταση του αντικειμένου.
  - Τη μέθοδο equals η οποία ελέγχει για ισότητα δύο αντικειμένων

- Και οι δύο συναρτήσεις ορίζονται από τον προγραμματιστή
  - Το τι String θα επιστραφεί και τι σημαίνει δύο αντικείμενα να είναι ίσα μπορούν να οριστούν όπως μας βολεύει.

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 Strings
  - Καλείται αυτόματα η toString

- Η μέθοδος equals ορίζεται πάντα ως:

```
public boolean equals(<Class name> other){
    …
}
```

Παράδειγμα

- 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.

toString()

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

Μετά καλούμε τη συνάρτηση toString() της κλάσης Integer

Χρησιμοποιούμε τις myCar1,myCar2 σαν String. Καλείται η μέθοδος toString() αυτόματα

```
class Car
{
   private Integer position = 0;

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

   public void move(int delta){
      position += delta ;
   }

   public String toString(){
      return position.toString();
   }
}

class MovingCarToString
{
   public static void main(String args[]){
      Car myCar1 = new Car(1);
      Car myCar2 = new Car(0); myCar2.move(2);
      System.out.println("Car 1 is at " + myCar1 + " and car 2 is at " + myCar2);
   }
}
```

Ισοδύναμο με το:

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

toString()

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

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

  public void move(int delta){
    position += delta ;
  }

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

class MovingCarToString
{
  public static void main(String args[]){
    Car myCar1 = new Car(1);
    Car myCar2 = new Car(0); myCar2.move(2);
    System.out.println("Car 1 is at " + myCar1 + " and car 2 is at " + myCar2);
  }
}
```

Ένας άλλος τρόπος να μετατρέψουμε ένα int σε String

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

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

  public void move(int delta){
    position += delta ;
  }

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

class MovingCarEquals
{
  public static void main(String args[]){
    Car myCar1 = new Car(2);
    Car myCar2 = new Car(0); myCar2.move(2);
    if (myCar1.equals(myCar2)){
       System.out.println("Collision!");
    }
  }
}
```

Η Java περιμένει αυτό το συντακτικό για τον ορισμό της equals

Ένα παράδειγμα αντικειμένου ως παράμετρος συνάρτησης

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

Χρήση της return για έλεγχο ροής

Κλήση της equals στο πρόγραμμα

Παράδειγμα

- Πως θα ορίσουμε τις μεθόδους toString και equals για την κλάση Person?

```
class Person
{
    private String name;

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

    public String toString(){
        return name;
    }

    public boolean equals(Person other){
        return this.name.equals(other.name);
    }

}

public class TwoPersons
{
    public static void main(String[] args){
        Person alice = new Person("Alice");
        Person bob = new Person("Bob");
        if (!alice.equals(bob)){
            System.out.println("There are two different persons: "
                                                    + alice + "and " + bob);
        }
    }
}
```

```
class Person{
    private String firstName;
    private String lastName;

    public Person(String firstName, String lastName){
        this.firstName = firstName;
        this.lastName = lastName;
    }

    public String toString(){
        return firstName + " " + lastName;
    }

    public boolean equals(Person other){
        return (this.firstName.equals(other.firstName))
                && (this.lastName.equals(other.lastName));
    }

}

public class TwoPersons2
{
    public static void main(String[] args){
        Person alice = new Person("Alice", "Wonderland");
        Person bob = new Person("Bob", "Sfougkarakis");
        if (!alice.equals(bob)){
            System.out.println("There are two different persons: "
                                                    + alice + "and " + bob);
        }
    }
}
```

ΑΝΤΙΚΕΙΜΕΝΑ ΜΕΣΑ ΣΕ ΑΝΤΙΚΕΙΜΕΝΑ

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, String name){
    this.position = position;
    this.driver = new Person(name);
  }

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

```
class MovingCarDriver1
{
  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;

  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 MovingCarDriver2
{
  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;
  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 MovingCarDriver3
{
  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

```
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;
      }
      else{
         System.exit(-1);
      }
   }
}
```

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.

```
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

Φωλιασμένες toString και equals

- Ένα αντικείμενο μπορεί να περιέχει μέσα άλλα αντικείμενα.
- Είναι συνηθισμένο σε αυτή την περίπτωση η toString και equals να ορίζονται κάνοντας φωλιασμένη κλήση της toString και της equals των αντικειμένων που περιέχει.
- Φωλιασμένη κλήση της toString της κλάσης driver

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

- Φωλιασμένη κλήση της toString της κλάσης driver

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

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

ΑΝΤΙΚΕΙΜΕΝΑ ΩΣ ΕΠΙΣΤΡΕΦΟΜΕΝΕΣ ΤΙΜΕΣ

Αντικείμενα ως επιστρεφόμενες τιμές

- Μία 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 μετά.

```
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;
  }

  public Person getDriver(){
    return driver;
  }
}
```

Επιστρέφει το αντικείμενο Person το οποίο είναι ο οδηγός του οχήματος.

```
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.

Παράδειγμα

- Κάνετε μια μέθοδο για την κλάση Car η οποία να παίρνει σαν όρισμα ένα αριθμό Κ και να κάνει Κ κινήσεις με τυχαίο αριθμό από βήματα στο διάστημα μεταξύ 0 έως 9

```
import java.util.Random;

class Car
{
	private int position = 0;

	public int[] move(int K){
		int[] moves = new int[K];
		Random rnd = new Random();
		for (int i = 0; i < K; i ++){
			int delta = rnd.nextInt(10);
			position += delta ;
			moves[i] = delta;
		}
		return moves;
	}

	public void printPosition(){
		System.out.println("Car is at position "+position);
	}

}

class MovingCar14
{
	public static void main(String args[]){
		Car myCar = new Car();
		int[] moves = myCar.move(4);
		myCar.printPosition();
		System.out.print("moves: ");
		for (int i = 0; i < 4; i ++){
			System.out.print(moves[i]+" ");
		}
		System.out.println();
	}
}
```