9. ΑΝΤΙΚΕΙΜΕΝΑ ΩΣ ΟΡΙΣΜΑΤΑ

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, 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;

  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;
  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

```
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

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.

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.

10. ΑΝΤΙΚΕΙΜΕΝΑ ΜΕ ΠΙΝΑΚΕΣ. CONSTRUCTORS. ΥΛΟΠΟΙΗΣΗ ΣΤΟΙΒΑΣ

Ένα *ιστόγραμμα* τιμών μετράει για ένα σύνολο από τιμές πόσες φορές εμφανίστηκε η κάθε τιμή. Για παράδειγμα αν έχω τις τιμές: 1,2,1,2,4,5,3,3,3,2,4 το ιστόγραμμα τους είναι 2,3,3,2,1, και είναι ο αριθμός εμφανίσεων των τιμών 1,2,3,4,5 αντίστοιχα (η τιμή 1 εμφανίζεται 2 φορές, η τιμή 2, 3 φορές, κοκ).

Στην άσκηση αυτή θα υλοποιήσετε μια κλάση **GradeHistogram** η οποία κρατάει ένα ιστόγραμμα για τους βαθμούς ενός μαθήματος. Η κλάση σας θα πρέπει να κρατάει το μέγιστο βαθμό για το μάθημα, και ένα πίνακα τον αριθμό εμφανίσεων του κάθε βαθμού. Αν ο μέγιστος βαθμός είναι maxGrade τότε οι πιθανοί βαθμοί θα είναι όλοι οι ακέραιοι στο διάστημα [1,maxGrade]. Η κλάση θα πρέπει να έχει και τις εξής μεθόδους:

1. Ένα **constructor,** ο οποίος θα παίρνει σαν όρισμα τον μέγιστο βαθμό και ένα πίνακα με βαθμούς και δημιουργεί το ιστόγραμμα των βαθμών.
2. Μια μέθοδο **toString**, η οποία θα επιστρέφει ένα String που αναπαριστά το ιστόγραμμα. Για το ιστόγραμμα στο παραπάνω παράδειγμα θα επιστρέφει το String: «1:2 2:3 3:3 4:2 5:1».
3. Την μέθοδο **equals**, η οποία θα συγκρίνει αν δύο ιστογράμματα είναι ίδια.
4. Μια μέθοδο **addHistogram** η οποία παίρνει σαν όρισμα ένα άλλο ιστόγραμμα (ένα αντικείμενο τύπου **GradeHistogram**) και, εφόσον έχουν τον ίδιο μέγιστο βαθμό, το προσθέτει στο υπάρχον ιστόγραμμα.

Σας δίνεται η κλάση **GradeHistogramTest,** για να τεστάρετε την κλάση σας. Όταν υλοποιήσετε τις μεθόδους που καλούνται στην main, βγάλετε τα σχόλια από τις αντίστοιχες εντολές για να τεστάρετε τις μεθόδους. Τεστάρετε τον κώδικα σας σταδιακά όπως φαίνεται στα σχόλια.

Μαθήματα από το lab

- Τι πληροφορία (δεδομένα) θέλουμε να κρατάει η κλάση μας?
  - Το μέγιστο βαθμό
  - Τις τιμές του ιστογράμματος

- Η πληροφορία (τα δεδομένα) που θέλουμε να κρατάει η κλάση θα είναι τα πεδία της κλάσης
  - Έναν ακέραιο maxGrade με το μέγιστο βαθμό που θα είναι και το μήκος του πίνακα
  - Ένα πίνακα ακεραίων histogram με τις συχνότητες για τον κάθε βαθμό

Κατασκευή ιστογράμματος

- Πως φτιάχνουμε ένα ιστόγραμμα από ένα πίνακα με βαθμούς?
  - Κάθε φορά που βλέπουμε τον βαθμό x θα πρέπει να αυξήσουμε την x-θέση του ιστογράμματος κατά ένα.

```
for (int i = 0; i < grades.length; i ++){
     int x = grades[i];
     histogram[x-1] ++;
}
```

Η μέθοδος toString

- Στην μέθοδο toString πρέπει να δημιουργήσουμε το String το οποίο θα αναπαριστά το ιστόγραμμα. Για να το κάνουμε αυτό θα πρέπει να διατρέξουμε τον πίνακα με τις τιμές και να φτιάξουμε το String αυξητικά.

Η μέθοδος toString ορίζεται πάντα έτσι

```
public String toString(){
    String output = "";
    for (int i = 0; i < maxGrade; i ++){
       output = output + (i+1) + ":" + histogram[i] + " ";
    }
    return output;
}
```

```
class GradeHistogram
{

  public GradeHistogram(int maxGrade, int[] grades)
  {
        int[] histogram = new int[maxGrade];
        for (int i = 0; i < grades.length; i ++){
              int x = grades[i];
              histogram[x-1] ++;
        }
  }

  public String toString(){
  {
        String output = "";
        for (int i = 0; i < maxGrade; i ++){
              output = output + (i+1) +
                    ":" + histogram[i] + " ";
        }
        return output;
  }
}
```

Σωστό ή λάθος?

Οι μεταβλητές maxGrade και histogram 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 GradeHistogram
{
  private int maxGrade;
  private int[] histogram;

  public GradeHistogram(int maxGrade, int[] grades)
  {
        int[] histogram = new int[maxGrade];
        for (int i = 0; i < grades.length; i ++){
              int x = grades[i];
              histogram[x-1] ++;
        }
  }

  public String toString(){
  {
        String output = "";
        for (int i = 0; i < maxGrade; i ++){
              output = output + (i+1) +
                    ":" + histogram[i] + " ";
        }
        return output;
  }
}
```

Σωστό?

Ο constructor δεν αρχικοποιεί τα πεδία της κλάσης .

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

ΛΑΘΟΣ!

```
class GradeHistogram
{
  private int maxGrade;
  private int[] histogram;

  public GradeHistogram(int maxGrade, int[]
  {
        this.maxGrade = maxGrade;
        for (int i = 0; i < grades.length; i ++){
              int x = grades[i];
              histogram[x-1] ++;
        }
  }

  public String toString(){
  {
        String output = "";
        for (int i = 0; i < maxGrade; i ++){
              output = output + (i+1) +
                    ":" + histogram[i] + " ";
        }
        return output;
  }
}
```

Σωστό?

Η μεταβλητή maxGrade αρχικοποιείται σωστά.

Ο πίνακας histogram όμως όχι.

Τον έχουμε ορίσει σωστά αλλά δεν τον έχουμε δημιουργήσει (δεν του έχουμε δώσει χώρο)! Δεν έχουμε προσδιορίσει το μέγεθος του

ΛΑΘΟΣ!

```
class GradeHistogram
{
   private int maxGrade;
   private int[] histogram = new int[maxGrade];

   public GradeHistogram(int maxGrade, int[] gra
   {
          this.maxGrade = maxGrade;
          for (int i = 0; i < grades.length; i ++){
                 int x = grades[i];
                 histogram[x-1] ++;
          }
   }

   public String toString(){
   {
          String output = "";
          for (int i = 0; i < maxGrade; i ++){
                 output = output + (i+1) +
                        ":" + histogram[i] + " ";
          }
          return output;
   }
}
```

Σωστό?

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

ΛΑΘΟΣ!

```
class GradeHistogram
{
  private int maxGrade;
  private int[] histogram;

  public GradeHistogram(int maxGrade, int[] grade
  {
        histogram = new int[maxGrade];
        for (int i = 0; i < grades.length; i ++){
              int x = grades[i];
              histogram[x-1] ++;
        }
  }

  public String toString(){
  {
        String output = "";
        for (int i = 0; i < maxGrade; i ++){
              output = output + (i+1) +
                    ":" + histogram[i] + " ";
        }
        return output;
  }
}
```

Σωστό?

Ο Constructor θα αρχικοποιήσει σωστά τον πίνακα histogram, αλλά δεν θα αλλάξει το πεδίο maxGrade μιας και χρησιμοποιεί την τοπική μεταβλητή - παράμετρο

Το maxGrade εδώ αναφέρεται στο πεδίο και έχει τιμή μηδέν.

ΛΑΘΟΣ!

```
class GradeHistogram
{
   private int maxGrade;
   private int[] histogram;

   public GradeHistogram(int maxGrade, int[] gra
   {
          this.maxGrade = maxGrade;
          histogram = new int[maxGrade];
          for (int i = 0; i < grades.length; i ++){
                 int x = grades[i];
                 histogram[x-1] ++;
          }
   }

   public String toString()
   {
          String output = "";
          for (int i = 0; i < maxGrade; i ++){
                 output = output + (i+1) +
                        ":" + histogram[i] + " ";
          }
          return output;
   }
}
```

Σωστό?

Πρώτα δηλώνουμε τα πεδία μέσα στην κλάση

Στον Constructor δίνουμε τιμή στο maxGrade και αφού πλέον ξέρουμε το μήκος του πίνακα τον δημιουργούμε και του δίνουμε χώρο για να κρατάει τις τιμές.

Τώρα μπορούμε και να κάνουμε και την αρχικοποίηση του πίνακα

ΣΩΣΤΟ!

Ορισμός και δημιουργία μεταβλητών

- Τι σημαίνει ορίζω μια μεταβλητή?
  - Οπουδήποτε έχουμε κώδικα της μορφής

    <τυπος> <όνομα μεταβλητής>

    ορίζουμε μια καινούρια μεταβλητή με αυτό το όνομα. Π.χ.,
    - `int maxGrade`
    - `int[] histogram`
    - `GradeHistogram hist`
- Τι σημαίνει δημιουργώ μια μεταβλητή/αντικείμενο
  - Δημιουργώ σημαίνει ότι δίνω χώρο στην μνήμη και αυτό γίνεται με την new. Χωρίς την κλήση της new το αντικείμενο δεν υπάρχει. Εξαίρεση, οι βασικοί τύποι (int, double, boolean).
    - `histogram = new int[maxGrade];`
    - `hist = new GradeHistogram(maxGrade, grades);`

Εμβέλεια μεταβλητών

- Η κάθε μεταβλητή έχει εμβέλεια μέσα στο block στο οποίο ορίζεται.
  - Τις μεταβλητές-πεδία της κλάσης μπορούν να τις χρησιμοποιήσουν όλες οι μέθοδοι της κλάσης
    - Οι μεταβλητές έχουν ζωή όσο υπάρχει το αντίστοιχο αντικείμενο της κλάσης
  - Οι μεταβλητές που ορίζονται μέσα σε μία μέθοδο μπορούν να χρησιμοποιηθούν μόνο μέσα στη μέθοδο.
    - Οι μεταβλητές χάνονται όταν βγούμε από τη μέθοδο.
  - Οι παράμετροι μιας μεθόδου είναι σαν τοπικές μεταβλητές της μεθόδου.

Παράδειγμα

```
class GradeHistogram
{
  private int maxGrade;
  private int[] histogram;

  public GradeHistogram(int maxGrade, int[] grades)
  {
        this.maxGrade = maxGrade;
        histogram = new int[maxGrade];
        for (int i = 0; i < grades.length; i ++){
              int x = grades[i];
              histogram[x-1] ++;
        }
  }
}
```

Ορισμός μεταβλητής πίνακα

Δημιουργία πίνακα

Οι κόκκινες μεταβλητές υπάρχουν μόνο μέσα στο μπλοκ της μεθόδου

Οι μπλε μεταβλητές είναι πεδία

Παράδειγμα (λάθος)

```
class GradeHistogram
{
  private int maxGrade;
  private int[] histogram;

  public GradeHistogram(int maxGrade, int[] grades)
  {
      this.maxGrade = maxGrade;
      int[] histogram = new int[maxGrade];
      for (int i = 0; i < grades.length; i ++){
            int x = grades[i];
            histogram[x-1] ++;
      }
  }
}
```

Το πεδίο-πίνακας **histogram** έχει οριστεί αλλά δεν έχει δημιουργηθεί

Ορισμός τοπικής μεταβλητής και δημιουργία της

Η τοπική μεταβλητή **histogram** χάνεται μόλις βγούμε από τον constructor

Οι κόκκινες μεταβλητές υπάρχουν μόνο μέσα στο μπλοκ της μεθόδου

Οι μπλε μεταβλητές είναι πεδία

Η μέθοδος equals

Η μέθοδος equals ορίζεται πάντα έτσι

```
public boolean equals(GradeHistogram other)
{
      if (this.maxGrade != other.maxGrade){
            return false;
      }
      for (int i = 0; i < maxGrade; i ++){
            if (this.histogram[i] != other.histogram[i]){
                  return false;
            }
      }
      return true;
 }
```

Είναι πιο εύκολο να ελέγξουμε για την περίπτωση της ανισότητας παρά της ισότητας. Μόλις μία από τις συνθήκες δεν ικανοποιείται επιστρέφουμε false. Αν φτάσουμε μέχρι τέλους ικανοποιούνται όλες και άρα επιστρέφουμε true

Δεν κάνουμε έλεγχο ισότητας χρησιμοποιώντας την toString!
Το String που επιστρέφουμε μπορεί να μην ικανοποιεί τον έλεγχο ισότητας

Η μέθοδος addHistogram

Η μέθοδος δεν επιστρέφει κάτι μιας και το αποτέλεσμα της πρόσθεσης θα αποθηκευτεί στο αντικείμενο

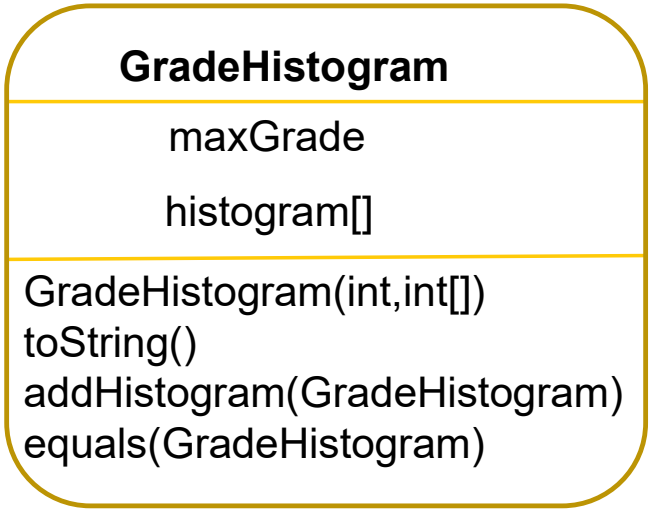
Η μέθοδος παίρνει σαν όρισμα ένα αντικείμενο GradeHistogram το οποίο θα προσθέσει

```
public void addHistogram(GradeHistogram other)
{
      if (this.maxGrade != other.maxGrade){
            return;
      }
      for (int i=0; i < maxGrade; i++){
        this.histogram[i] += other.histogam[i];
      }
}
```

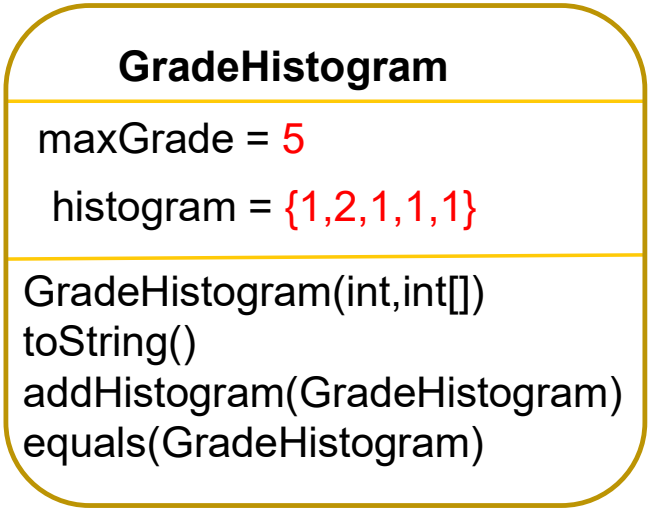
Έχουμε πρόσβαση στα πεδία του other γιατί είναι της ίδιας κλάσης με το αντικείμενο που καλεί την addHistogram

Κλάσεις και αντικείμενα

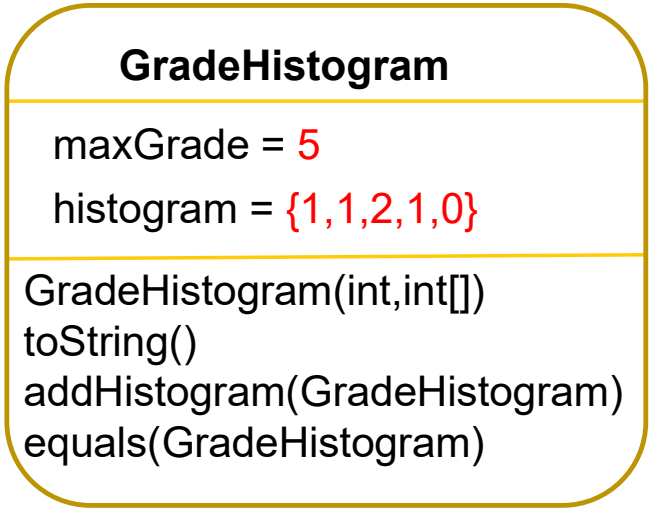
Ορισμός της κλάσης



hist2 =
new GradeHistogram(5,grades2)

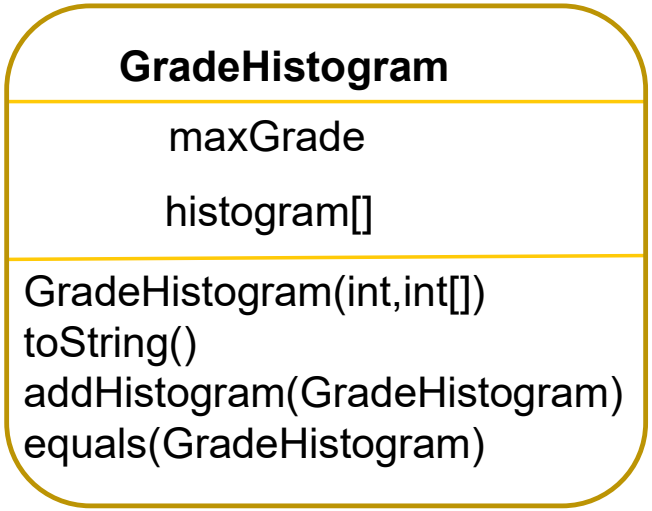


hist3 =
new GradeHistogram(5,grades3)



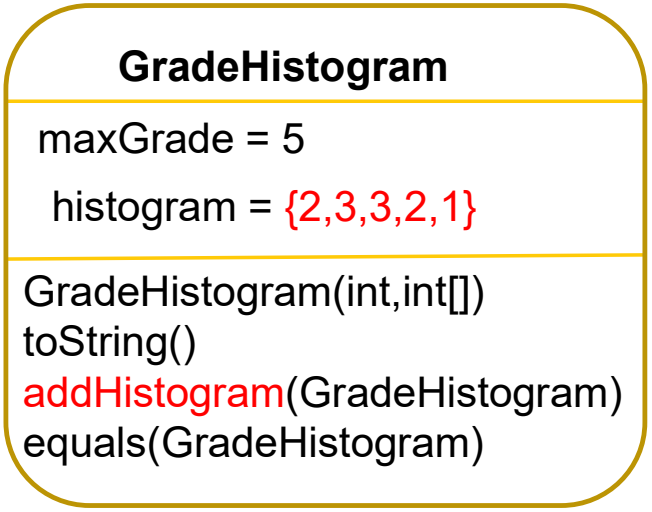
Κλάσεις και αντικείμενα

Ορισμός της κλάσης

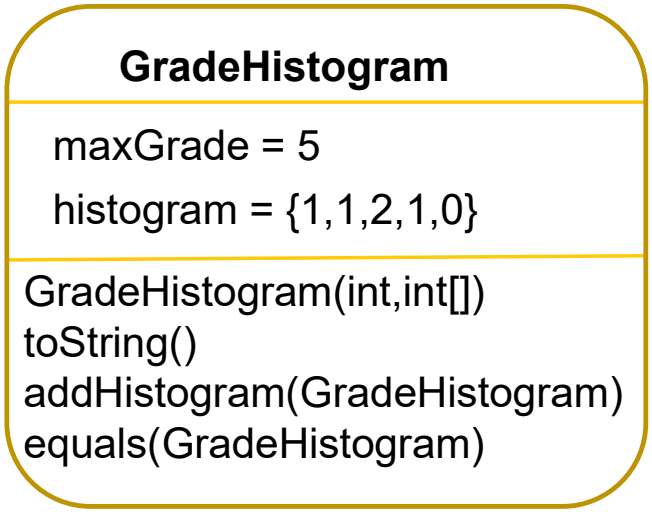


hist2.addHistogram(hist3);

hist2 =
new GradeHistogram(5,grades2)



hist3 =
new GradeHistogram(5,grades3)



```
class GradeHistogram
{
  private int maxGrade;
  private int[] histogram;
  private String output = "";

  public Geometric(int maxGrade, int[] grades)
  {
        this.maxGrade = maxGrade;
        histogram = new int[maxGrade];
        for (int i = 0; i < grades.length; i ++){
              x = grades[i];
              histogram[x-1] ++;
        }
  }

  public String toString(){
  {
        for (int i = 0; i < maxGrade; i ++){
              output = output + (i+1) +
                    ":" + histogram[i] + " ";
        }
        return output;
  }
}
```

Σωστό?

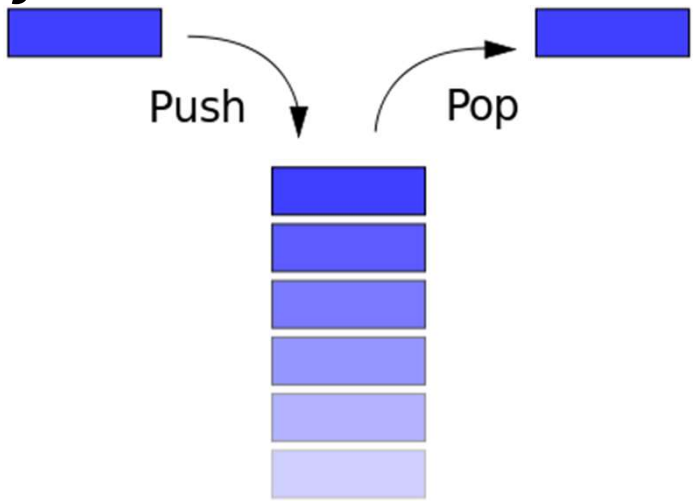
Η μεταβλητή output πλέον είναι πεδίο. Οι αλλαγές της τιμής της παραμένουν στο αντικείμενο

Τι γίνεται αν κάνουμε πολλαπλές κλήσεις της μεθόδου toString?

ΛΑΘΟΣ!

Παράδειγμα ADT: Στοίβα (Stack)

- Η Στοίβα είναι μια συλλογή δεδομένων η οποία επιτρέπει τις εξής λειτουργίες:
  - push(element): προσθέτει ένα νέο στοιχείο στην κορυφή της στοίβας
  - pop(): αφαιρεί και επιστρέφει το στοιχείο το οποίο βρίσκεται στην κορυφή της στοίβας.
  - isEmpty(): ελέγχει αν η στοίβα είναι άδεια και επιστρέφει true ή false
- Η Στοίβα υλοποιεί την πολιτική Last-In-First-Out (LIFO) στη σειρά που μας δίνει τα στοιχεία
  - Χρήσιμο σε διάφορες εφαρμογές, π.χ., για τη δέσμευση μνήμης στην κλήση συναρτήσεων



Υλοποίηση

- Θα υλοποιήσουμε μια Στοίβα ακεραίων χρησιμοποιώντας ένα πίνακα (Στοιβα συγκεκριμένης χωρητικότητας)
  - Τι πεδία πρέπει να ορίσουμε?
  - Τι μεθόδους?

```
class Stack
{
            private int capacity;
            private int size = 0;
            private int[] elements;

            public Stack(int capacity){
                        this.capacity = capacity;
                        elements = new int[capacity];
            }

            public void push(int element){
                        if (size == capacity){
                                    System.out.println("Cannot enter any more elements");
                                    return;
                        }
                        elements[size] = element;
                        size ++;
            }

            public int pop(){
                        if (size == 0){
                                    System.out.println("No elements to pop");
                                    return -1;
                        }
                        size -- ;
                        return elements[size];
            }

            public boolean isEmpty(){
                        return (size == 0);
            }
}
```

Εφαρμογές

- Υπολόγισε την δυαδική μορφή ενός ακεραίου.

```
class Binary
{
	public static void main(String[] args)
	{
		Stack myStack = new Stack(100);
		int number = 1973;

		while (number > 0){
			myStack.push(number%2);
			number = number/2;
		}

		while (!myStack.isEmpty()){
			System.out.print(myStack.pop());
		}
	}
}
```

Επεκτάσεις

- Πως θα ορίσουμε την μέθοδο equals?
- Πως θα ορίσουμε τη μέθοδο toString?

```
public String toString(){
	String returnString = "":
	for (int i = 0; i < size; i ++){
		returnString = returnString + elements[i] + " ";
	}
	return returnString;
}

public boolean equals(Stack other){
	if (this.size != other.size){
		return false;
	}
	for (int i = 0; i < size; i ++){
		if (this.elements[i] != other.elements[i]){
			return false;
		}
	}
	return true;
}
```

11. ΑΝΑΦΟΡΕΣ Ι

Στοίβα και σωρός

Αναφορές-παράμετροι

String Interning

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;
```

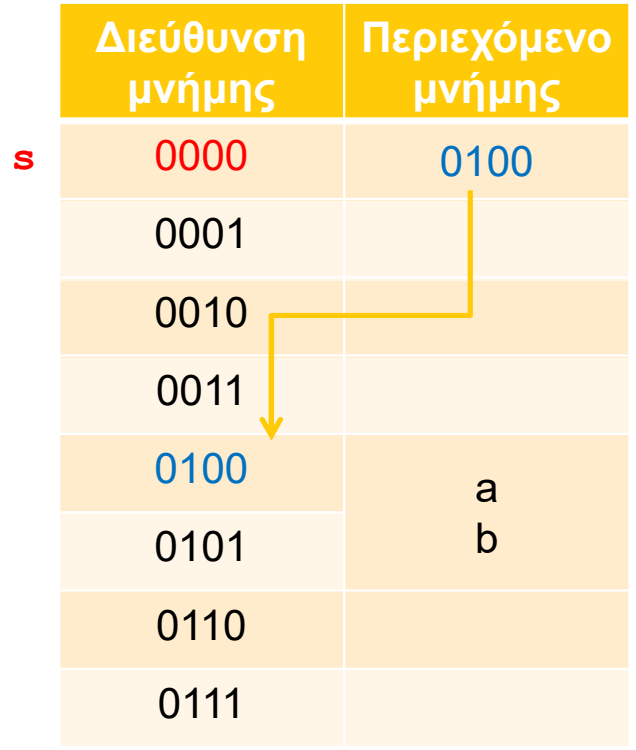
μας λέει ότι έχουμε ένα πίνακα από ακέραιους αλλά δεν μας λέει πόσο μεγάλος θα είναι αυτός ο πίνακας.

```
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[0] = 10;
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[0] = 10;
A = new int[3];
```

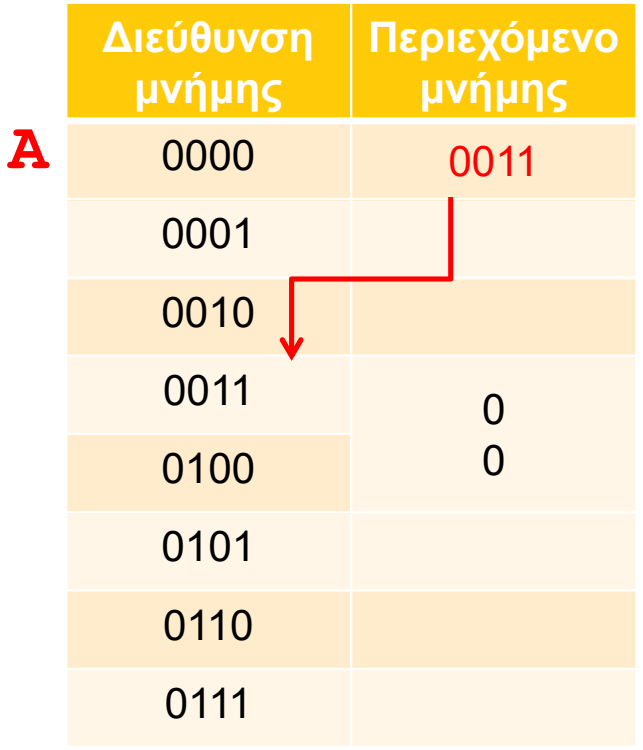
Η δεσμευμένη λέξη null σημαίνει μια κενή αναφορά (μια διεύθυνση που δεν δείχνει πουθενά)

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

Παράδειγμα - πινάκες

```
int[] A;
A = new int[2];
A[0] = 10;
A = new int[3];
```

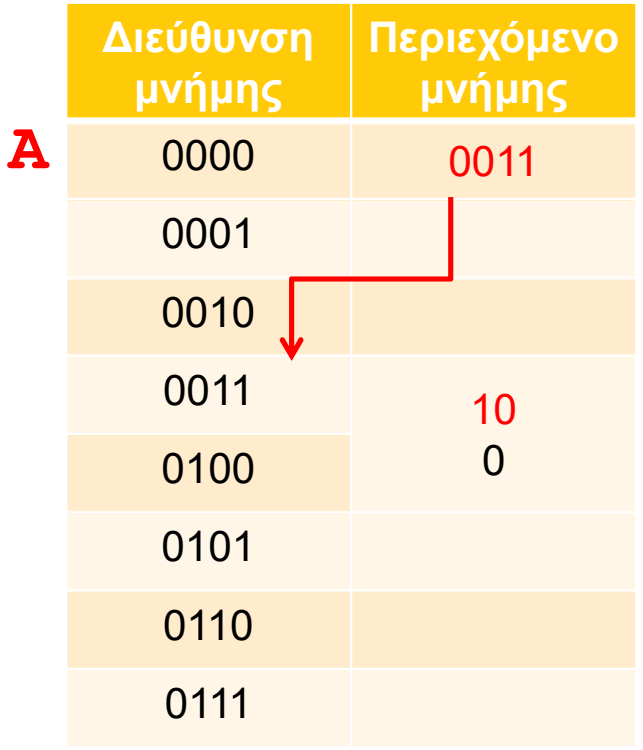
Με την εντολή new δεσμεύουμε δύο θέσεις ακεραίων και η αναφορά του A δείχνει σε αυτό το χώρο που δεσμεύσαμε



Παράδειγμα - πινάκες

```
int[] A;
A = new int[2];
A[0] = 10;
A = new int[3];
```

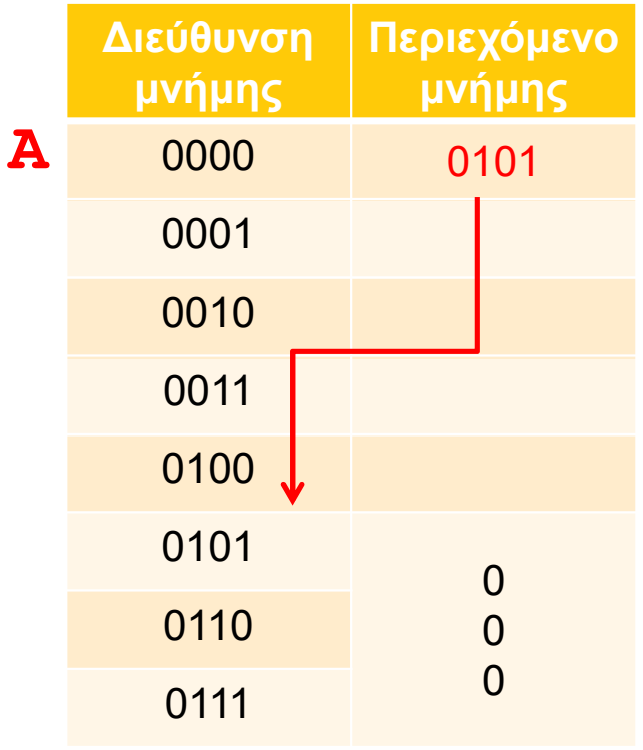
Ο τελεστής [] για τον πίνακα μας πάει στην αντίστοιχη θέση του χώρου που κρατήσαμε



Παράδειγμα - πινάκες

```
int[] A;
A = new int[2];
A[0] = 10;
A = new int[3];
```

Με νέα κλήση της new δεσμεύουμε νέο χώρο για το A, και αν δεν έχουμε κρατήσει την προηγούμενη αναφορά σε κάποια άλλη μεταβλητή τότε χάνεται (garbage collection), όπως και οι τιμές που είχαμε αποθηκεύσει στον πίνακα.



Αντικείμενα κλάσεων

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

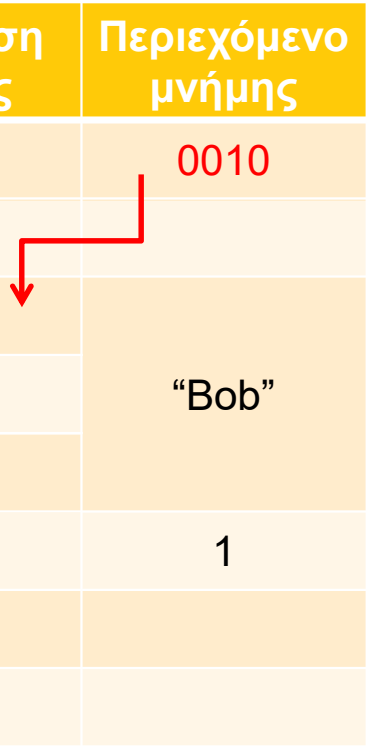
```
public class Person
{
    private String name;
    private int number;

    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);
```

Η κλήση της new δημιουργεί ένα χώρο μνήμης για την αποθήκευση του αντικειμένου τύπου Person το οποίο κρατάει ένα string και ένα ακέραιο (δεσμεύεται χώρος και γι αυτά).

Η μεταβλητή `varP` κρατάει την διεύθυνση του χώρου στην μνήμη όπου αποθηκεύσαμε αυτό το αντικείμενο.



Αναθέσεις μεταξύ αντικειμένων

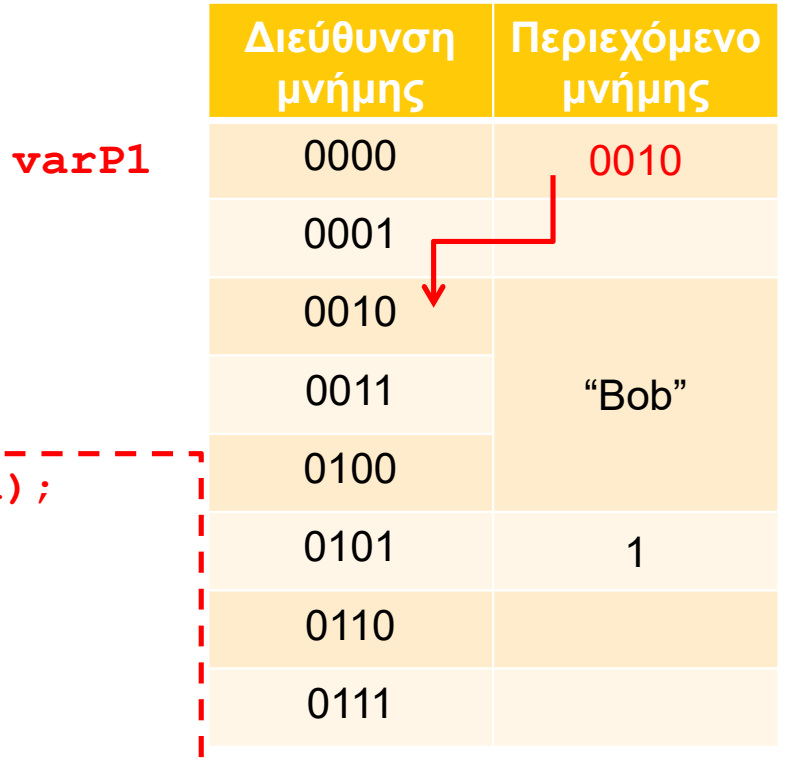
Τι θα τυπώσει το παρακάτω πρόγραμμα?

```
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	

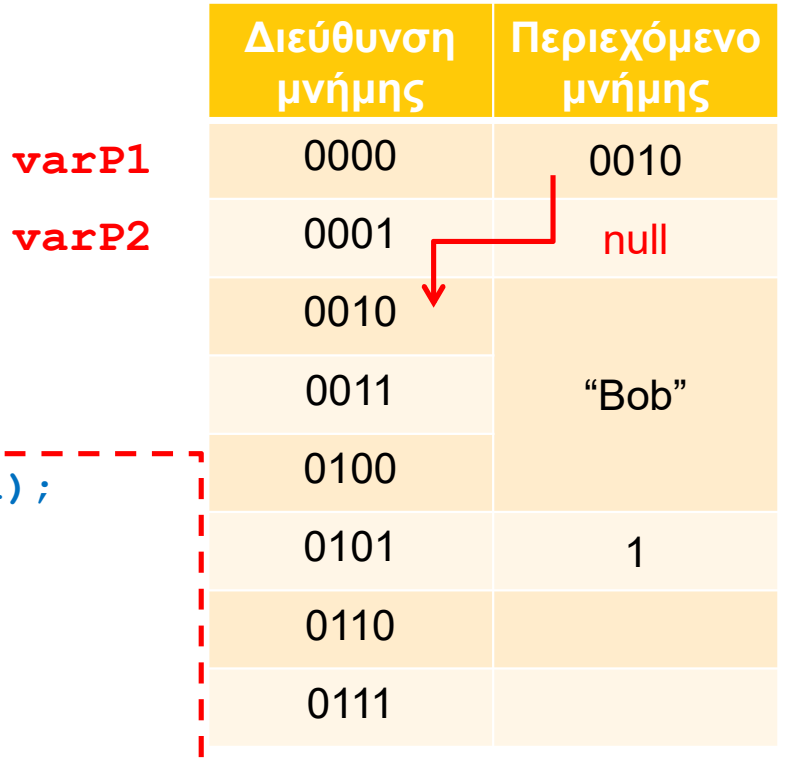
Αναθέσεις μεταξύ αντικειμένων

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



Αναθέσεις μεταξύ αντικειμένων

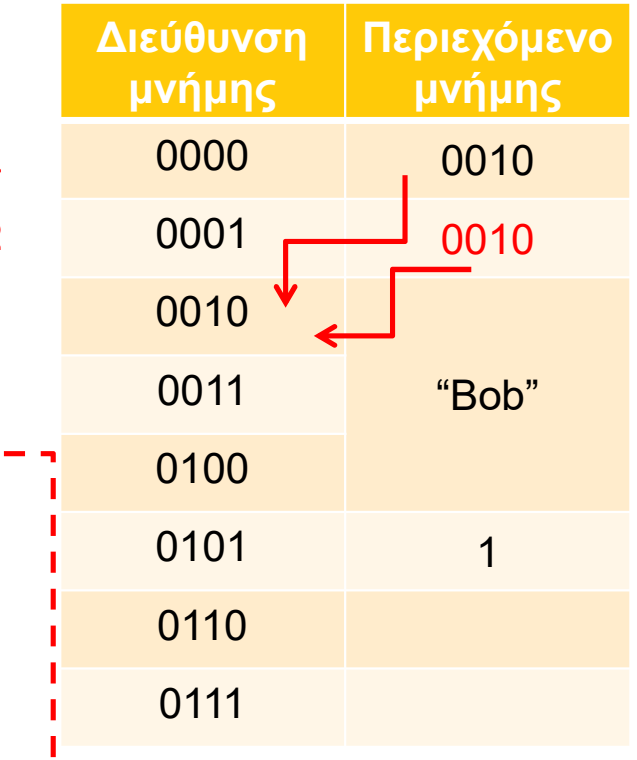
```
Person varP1 = new Person("Bob", 1);
Person varP2;
varP2 = varP1;
varP2.set("Ann",2);
System.out.println(varP1);
```



Αναθέσεις μεταξύ αντικειμένων

Η ανάθεση του `varP1` στο `varP2` έχει αποτέλεσμα η μεταβλητή `varP2` να δείχνει στην ίδια θέση μνήμης όπως και η `varP1`

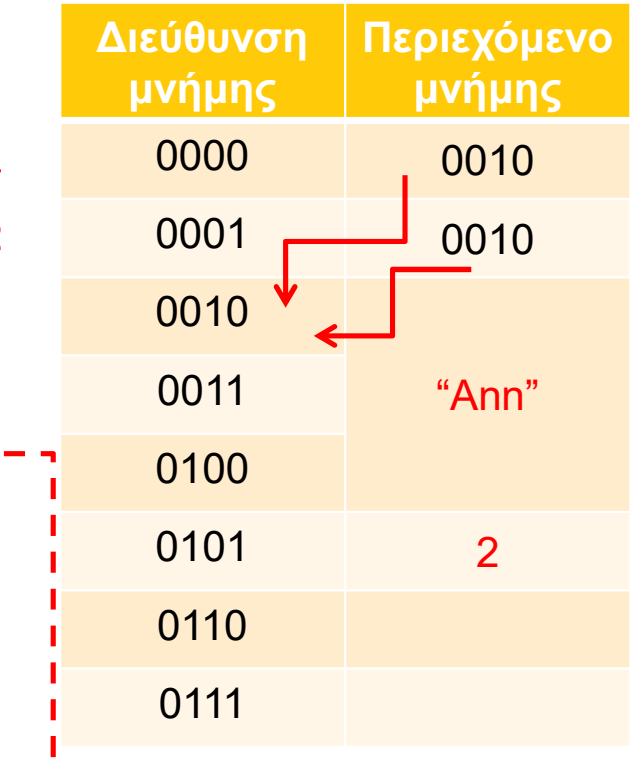
```
Person varP1 = new Person("Bob", 1);
Person varP2;
varP2 = varP1;
varP2.set("Ann",2);
System.out.println(varP1);
```



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**

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

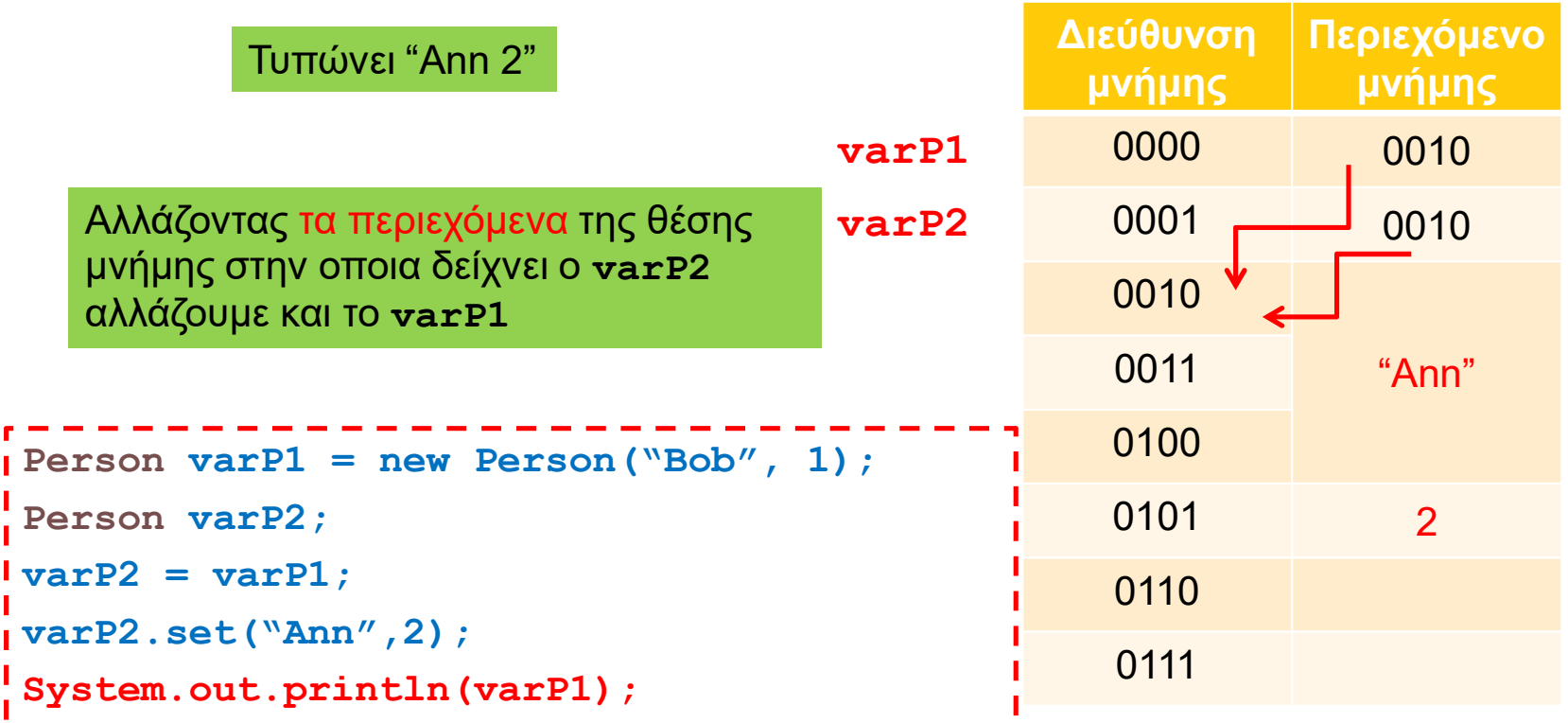


Αναθέσεις μεταξύ αντικειμένων

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);
```



15. ΕΙΣΑΓΩΓΗ ΣΤΗΝ ΚΛΗΡΟΝΟΜΙΚΟΤΗΤΑ

Παράδειγμα

- Στο παράδειγμα με το τμήμα πανεπιστημίου οι φοιτητές και οι καθηγητές είχαν κάποια κοινά στοιχεία
  - Και οι δύο είχαν όνομα
  - Και οι δύο είχαν κάποιο χαρακτηριστικό αριθμό
- και κάποιες διαφορές
  - Οι καθηγητές δίδασκαν μαθήματα
  - Οι φοιτητές έπαιρναν μαθήματα, βαθμούς και μονάδες

- Δεν θα ήταν βολικό αν είχαμε μεθόδους που να χειρίζονταν με κοινό τρόπο τις ομοιότητες (π.χ. εκτύπωση των βασικών στοιχείων) και να έχουν ξεχωριστές μεθόδους για τις διαφορές?
  - Έτσι δεν θα έπρεπε να γράφουμε τον ίδιο κώδικα πολλές φορές και οι αλλαγές θα έπρεπε να γίνουν μόνο μια φορά.

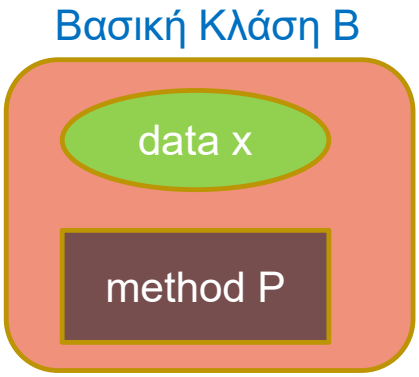
- Αυτό το καταφέρνουμε με την κληρονομικότητα!

Κληρονομικότητα

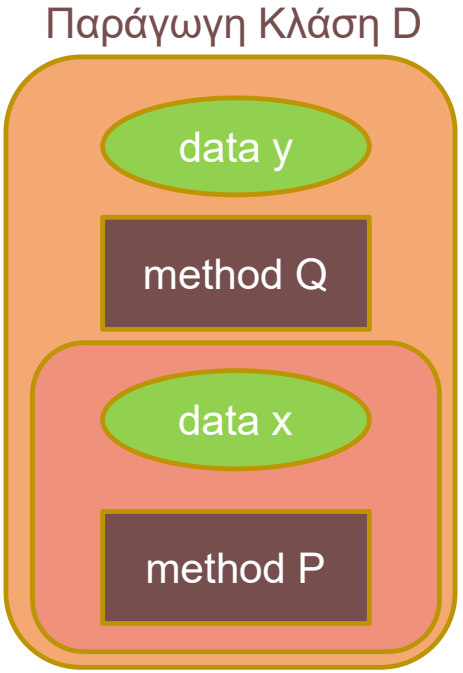
- Η κληρονομικότητα είναι κεντρική έννοια στον αντικειμενοστραφή προγραμματισμό.
- Η ιδέα είναι να ορίσουμε μια γενική κλάση που έχει κάποια χαρακτηριστικά (πεδία και μεθόδους) που θέλουμε και μετά να ορίσουμε εξειδικευμένες παραλλαγές της κλάσης αυτής στις οποίες προσθέτουμε ειδικότερα χαρακτηριστικά.
  - Οι εξειδικευμένες κλάσεις λέμε ότι κληρονομούν τα χαρακτηριστικά της γενικής κλάσης

Κληρονομικότητα

Έχουμε μια Βασική Κλάση (Base Class) Β, με κάποια πεδία και μεθόδους.



Θέλουμε να δημιουργήσουμε μια νέα κλάση D η οποία να έχει όλα τα χαρακτηριστικά της Β, αλλά και κάποια επιπλέον.



Αντί να ξαναγράψουμε τον ίδιο κώδικα δημιουργούμε μια Παράγωγη Κλάση (Derived Class) D, η οποία κληρονομεί όλη τη λειτουργικότητα της Βασικής Κλάσης Β και στην οποία προσθέτουμε τα νέα πεδία και μεθόδους.

Αυτή διαδικασία λέγεται κληρονομικότητα

Κληρονομικότητα

- Η κληρονομικότητα είναι χρήσιμη όταν
  - Θέλουμε να έχουμε αντικείμενα και της κλάσης B και της κλάσης D.
  - Θέλουμε να ορίσουμε πολλαπλές παράγωγες κλάσεις D1, D2, … που η κάθε μία επεκτείνει την B με διαφορετικό τρόπο.

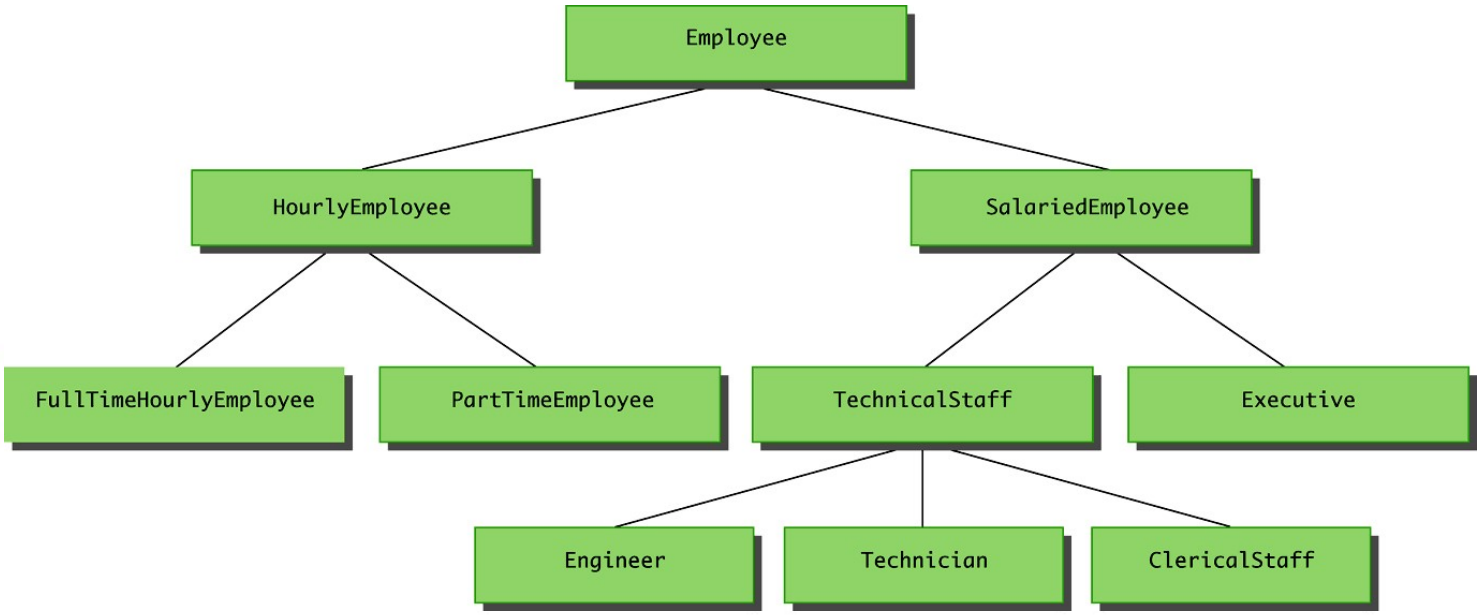
- Μπορούμε να ορίσουμε παράγωγες κλάσεις των παράγωγων κλάσεων.
  - Με αυτό τον τρόπο ορίζεται μια ιεραρχία κλάσεων.

Ιεραρχία κλάσεων (Class Hierarchy)

- Παράδειγμα: Έχουμε ένα πρόγραμμα που διαχειρίζεται τους Εργαζόμενους μιας εταιρίας.
  - Όλοι οι εργαζόμενοι έχουν κοινά χαρακτηριστικά το όνομα τους και το ΑΦΜ τους.
- Οι εργαζόμενοι χωρίζονται σε Ωρομίσθιους και Έμμισθους
  - Διαφορετικά χαρακτηριστικά θα κρατάμε όσον αφορά το μισθό για τον καθένα
- Οι Ωρομίσθιοι χωρίζονται σε Πλήρους και Μερικής απασχόλησης
- Οι Έμμισθοι χωρίζονται σε Τεχνικό Προσωπικό και Διευθυντικό προσωπικό
- Κ.ο.κ.…

A Class Hierarchy

Display 7.1 **A Class Hierarchy**

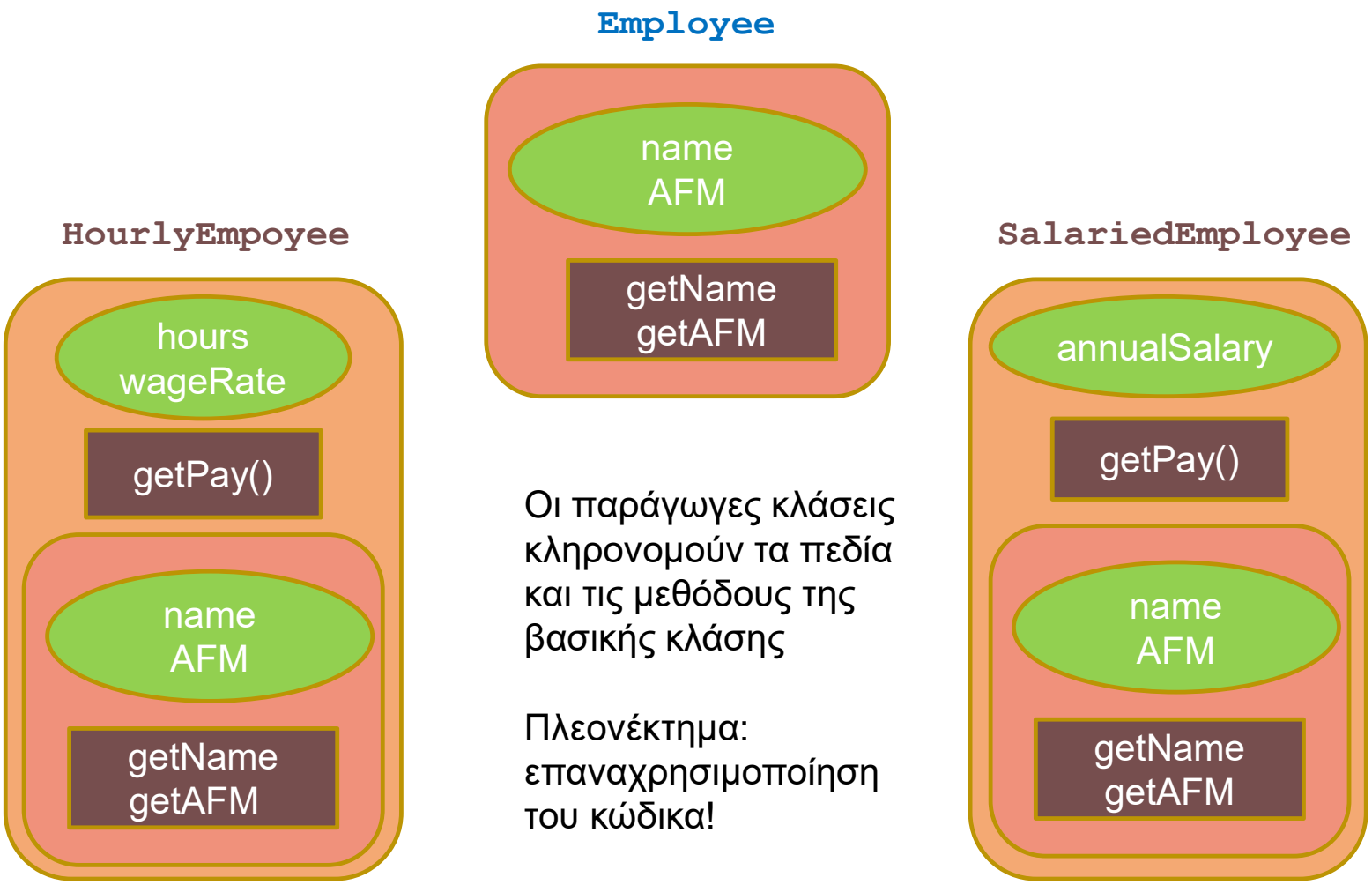


Ιεραρχία κλάσεων

- Η ιεραρχία από κλάσεις ορίζει κάτι σαν γενεαλογικό δέντρο κλάσεων από πιο γενικές προς πιο ειδικές κλάσεις.

- Στη Java όλες οι κλάσεις ανήκουν στην ίδια ιεραρχία.
  - Στην κορυφή της ιεραρχίας είναι η κλάση Object.

Παράδειγμα



Οι παράγωγες κλάσεις κληρονομούν τα πεδία και τις μεθόδους της βασικής κλάσης

Πλεονέκτημα: επαναχρησιμοποίηση του κώδικα!

Ορολογία

- Η βασική κλάση συχνά λέγεται και υπέρ-κλάση (superclass) και η παραγόμενη κλάση υπό-κλάση (subclass).
- Επίσης η βασική κλάση λέμε ότι είναι ο γονέας της παραγόμενης κλάσης, και η παράγωγη κλάση το παιδί της βασικής.
  - Αν έχουμε παραπάνω από ένα επίπεδο κληρονομικότητας στην ιεραρχία, τότε έχουμε πρόγονο και απόγονο κλάση.

Συντακτικό

- Ας πούμε ότι έχουμε την βασική κλάση **`Employee`** και τις παραγόμενες κλάσεις **`HourlyEmployee`** και **`SalariedEmployee`**.
- Για να ορίσουμε τις παραγόμενες κλάσεις χρησιμοποιούμε το εξής συντακτικό στη δήλωση της κλάσης:

- `public class HourlyEmployee extends Employee`
- `public class SalariedEmployee extends Employee`

Η βασική κλάση

```
public class Employee
{
    private String name;
    private int AFM;

    public Employee( ) { … }

    public Employee(String theName, int theAFM) { … }

    public Employee(Employee originalObject) { … }

    public String getName( ) { … }
    public void setName(String newName) { … }

    public int getAFM( ) { … }
    public void setAFM (int newAFM) { … }

    public String toString() { … }

}
```

Η παράγωγη κλάση HourlyEmployee

```
public class HourlyEmployee extends Employee
{
    private double wageRate;
    private double hours; //for the month

    public HourlyEmployee( ) { … }

    public HourlyEmployee(String theName, int theAFM,
                          double theWageRate, double theHours) { … }

    public HourlyEmployee(HourlyEmployee originalObject) { … }

    public double getRate( ) { … }
    public void setRate(double newWageRate) { … }

    public double getHours( ) { … }
    public void setHours(double hoursWorked) { … }

    public double getPay( ) {
        return wageRate*hours;
    }

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

Νέα πεδία για την HourlyEmployee

Μέθοδος getPay υπολογίζει το μηνιαίο μισθό

Η παράγωγη κλάση SalariedEmployee

```
public class SalariedEmployee extends Employee
{
    private double salary; //annual

    public SalariedEmployee( ) { … }

    public SalariedEmployee(String theName,
                            int theAFM, double theSalary) { … }

    public SalariedEmployee(SalariedEmployee originalObject ) { … }

    public double getSalary( ) { … }
    public void setSalary(double newSalary) { … }

    public double getPay( )
    {
        return salary/12;
    }

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

Νέα πεδία για την SalariedEmployee

Μέθοδος getPay υπολογίζει το μηνιαίο μισθό. Διαφορετική από την προηγούμενη

```
public class Example1
{
    public static void main(String[] args)
    {
        HourlyEmployee alice = new HourlyEmployee("Alice",
                                                  100, 50.5, 40);

        SalariedEmployee bob = new SalariedEmployee("Bob",
                                                    200, 100000);

        System.out.println("Alice: "
                           + alice.getName() + " "
                           + alice.getAFM() + " "
                           + alice.getPay());

        System.out.println("Bob: "
                           + bob.getName() + " "
                           + bob.getAFM() + " "
                           + bob.getPay());

    }
}
```

Μέθοδοι της Employee

Μέθοδοι των παράγωγων κλάσεων

Constructor

```
public class Employee
{
    private String name;
    private int AFM;


    public Employee()
    {
        name = "no name";
        AFM = 0;
    }

    public Employee(String theName, int theAFM)
    {
        if (theName == null || theAFM <= 0)
        {
            System.out.println("Fatal Error creating employee.");
            System.exit(0);
        }
        name = theName;
        AFM = theAFM;
    }
}
```

```
public class HourlyEmployee extends Employee
{
    private double wageRate;
    private double hours; //for the month


    public HourlyEmployee(String theName, int theAFM,
                          double theWageRate, double theHours)
    {
        super(theName, theAFM);
        if ((theWageRate >= 0) && (theHours >= 0))
        {
            wageRate = theWageRate;
            hours = theHours;
        }
        else
        {
           System.out.println(
                     "Fatal Error: creating an illegal hourly employee.");
           System.exit(0);
        }
    }
}
```

Με τη λέξη κλειδί super αναφερόμαστε στην βασική κλάση.

Εδώ καλούμε τον constructor της Employee με ορίσματα το όνομα και το ΑΦΜ

Ο constructor super μπορεί να κληθεί μόνο στην αρχή της μεθόδου.

```
public class SalariedEmployee extends Employee
{
    private double salary; //annual

    public SalariedEmployee(String theName,
                    int theAFM, double theSalary)
    {
        super(theName, theAFM);
        if (theSalary >= 0)
            salary = theSalary;
        else
        {
            System.out.println(
                 "Fatal Error: Negative salary.");
            System.exit(0);
        }
    }
}
```

```
public class SalariedEmployee extends Employee
{
    private double salary; //annual

    public SalariedEmployee()
    {
        super();
        salary = 0;
    }
}
```

Καλεί τον default constructor της Employee

Η εντολή δεν είναι απαραίτητη σε αυτή την περίπτωση. Αν δεν έχουμε κάποια κλήση προς τον constructor της γονικής κλάσης, τότε καλείται εξ ορισμού ο default constructor της Employee.

```
public class SalariedEmployee extends Employee
{
    private double salary; //annual

    public SalariedEmployee(String theName,int theAFM)
    {
        salary = 0;
    }
}
```

Πως θα αρχικοποιηθεί το αντικείμενο στην περίπτωση που κληθεί αυτός ο constructor?

Εφόσον δεν καλούμε εμείς κάποιο constructor της γονικής κλάσης θα κληθεί ο default constructor ο οποίος θα αρχικοποιήσει το όνομα στο “no name” και το ΑΦΜ στο μηδέν.

```
public class SalariedEmployee extends Employee
{
    private double salary; //annual

    public SalariedEmployee(String theName,int theAFM)
    {
        super(theName, theAFM);
        salary = 0;
    }
}
```

Αν θέλουμε να αρχικοποιήσουμε το όνομα και το ΑΦΜ θα πρέπει να καλέσουμε τον αντίστοιχο constructor της γονικής κλάσης.

Constructor this

- Όπως καλείται ο constructor super της γονικής κλάσης μπορούμε να καλέσουμε και τον constructor this της ίδιας κλάσης.

```
public class SalariedEmployee extends Employee
{
    private double salary; //annual

    public SalariedEmployee(String theName, int theAFM, double theSalary)
    {
          super(theName, theAFM);
          if (theSalary >= 0)
              salary = theSalary;
          else{
             System.out.println("Fatal Error: Negative salary.");
              System.exit(0);
          }
    }

    public SalariedEmployee(){
          this("no name", 0, 0);
    }
}
```

Καλεί ένα άλλο constructor της ίδιας κλάσης

Γιατί να μην κάνουμε κάτι πιο απλό? Κατευθείαν ανάθεση των πεδίων

```
public class SalariedEmployee extends Employee
{
    private double salary; //annual

    public SalariedEmployee(String theName,
                            int theAFM, double theSalary)
    {
        name = theName;
        AFM = theAFM;
        salary = theSalary;
    }
}
```

ΛΑΘΟΣ!

Οι παραγόμενες κλάσεις δεν έχουν πρόσβαση στα private πεδία και τις private μεθόδους της βασικής κλάσεις.

Κληρονομικότητα και ενθυλάκωση

- Οι παραγόμενες κλάσεις κληρονομούν την πληροφορία που έχει και η γονική κλάση
  - Ένα αντικείμενο SalariedEmployee έχει πληροφορία για το όνομα και το ΑΦΜ του υπαλλήλου.

- Δεν έχουν όμως πρόσβαση να διαβάσουν και να αλλάξουν ότι είναι private μέσα στην γονική κλάση.
  - Στην περίπτωση του SalariedEmployee, δεν μπορούμε να αλλάξουμε ή να διαβάσουμε το όνομα. Θα πρέπει να χρησιμοποιήσουμε τις public μεθόδους setName, getName.
  - Για τον constructor πρέπει να καλέσουμε την super.
- Με αυτό τον τρόπο προστατεύουμε τα δεδομένα της γονικής κλάσης από κώδικα εκτός της κλάσης.

- Ο περιορισμός ισχύει και για μεθόδους που είναι private στην γονική κλάση.

```
public class Employee
{
      private void doSomething(){
            System.out.println("doSomething");
      }
}
```

```
public class SalariedEmployee extends Employee
{
      public void doSomethingMore(){
            doSomething();
            System.out.println("and more");
      }
}
```

ΛΑΘΟΣ!

Protected μέλη

- Οι παράγωγες κλάσεις έχουν πρόσβαση σε όλα τα public πεδία και μεθόδους της γενικής κλάσης.
- ΔΕΝ έχουν πρόσβαση στα private πεδία και μεθόδους.
  - Μόνο μέσω public μεθόδων set* και get*

- Protected: αν κάποια πεδία και μέθοδοι είναι protected μπορούν να τα δουν όλοι οι απόγονοι της κλάσης.
  - Το βιβλίο δεν το συνιστά.

- Package access: αν δεν προσδιορίσετε public, private, ή protected access τότε η default συμπεριφορά είναι ότι η μεταβλητή είναι προσβάσιμη από άλλες κλάσεις μέσα στο ίδιο πακέτο.

Employee

```
public class Employee
{
    private String name = "default";
    private Date hireDate = new Date(11,4,2013);


    public Employee()
    {
       System.out.println("empty contructor");
    }

    public Employee(String theName, Date theDate)
    {
       name = theName;
       hireDate = new Date(theDate);
    }
}
```

```
public class HourlyEmployee extends Employee
{
    private double wageRate;
    private double hours; //for the month


    public HourlyEmployee(String theName, Date theDate,
                          double theWageRate, double theHours)
    {
        name = theName;
        hireDate = new Date(theDate);
        wageRate = theWageRate;
        hours = theHours;
    }
}
```

Χτυπάει λάθος η πρόσβαση σε private πεδία.

Employee

```
public class Employee
{
    protected String name = "default";
    protected Date hireDate = new Date(11,4,2013);


    public Employee()
    {
       System.out.println("empty contructor");
    }

    public Employee(String theName, Date theDate)
    {
       name = theName;
       hireDate = new Date(theDate);
    }
}
```

```
public class HourlyEmployee extends Employee
{
    private double wageRate;
    private double hours; //for the month


    public HourlyEmployee(String theName, Date theDate,
                          double theWageRate, double theHours)
    {
        name = theName;
        Date = new (theDate);
        wageRate = theWageRate;
        hours = theHours;
    }
}
```

OK η πρόσβαση σε protected πεδία.

Υπέρβαση μεθόδων (method overriding)

- Μία μέθοδος που ορίζεται στην βασική κλάση μπορούμε να την ξανα-ορίσουμε στην παράγωγη κλάση με διαφορετικό τρόπο
  - Παράδειγμα: η μέθοδος `toString()`. Την ξανα-ορίζουμε για κάθε παραγόμενη κλάση ώστε να παράγει αυτό που θέλουμε
  - Αυτό λέγετε υπέρβαση της μεθόδου (method overriding).
- Η υπέρβαση των μεθόδων είναι διαφορετική από την υπερφόρτωση.
  - Στην υπερφόρτωση αλλάζουμε την υπογραφή της μεθόδου.
  - Εδώ έχουμε την ίδια υπογραφή, απλά αλλάζει ο ορισμός στην παραγόμενη κλάση.

```
public class Employee
{
    private String name;
    private int AFM;

    public Employee( ) { … }

    public Employee(String theName, int theAFM) { … }

    public Employee(Employee originalObject) { … }

    public String getName( ) { … }
    public void setName(String newName) { … }

    public Date getAFM( ) { … }
    public void setAFM(int AFM) { … }

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

}
```

```
public class HourlyEmployee extends Employee
{
    private double wageRate;
    private double hours; //for the month

    public HourlyEmployee( ) { … }

    public HourlyEmployee(String theName, int theAFM,
                          double theWageRate, double theHours) { … }

    public HourlyEmployee(HourlyEmployee originalObject) { … }

    public double getRate( ) { … }
    public void setRate(double newWageRate) { … }

    public double getHours( ) { … }
    public void setHours(double hoursWorked) { … }

    public double getPay( ) {
        return wageRate*hours;
    }

    public String toString( ){
       return (getName( ) + " " + getAFM( )
            + "\n$" + wageRate + " per hour for " + hours + " hours");
    }
}
```

```
public class SalariedEmployee extends Employee
{
    private double salary; //annual

    public SalariedEmployee( ) { … }

    public SalariedEmployee(String theName,
                        int theAFM, double theSalary) { … }

    public SalariedEmployee(SalariedEmployee originalObject ) { … }

    public double getSalary( ) { … }
    public void setSalary(double newSalary) { … }

    public double getPay( )
    {
        return salary/12;
    }

    public String toString( ) { … }
    {
        return (getName( ) + " " + getAFM( )
                                + "\n$" + salary + " per year");
    }
}
```

```
public class SalariedEmployee extends Employee
{
    private double salary; //annual

    public SalariedEmployee( ) { … }

    public SalariedEmployee(String theName,
                            int theAFM, double theSalary) { … }

    public SalariedEmployee(SalariedEmployee originalObject ) { … }

    public double getSalary( ) { … }
    public void setSalary(double newSalary) { … }

    public double getPay( )
    {
        return salary/12;
    }

    public String toString( ) { … }
    {
        return (super.toString( ) + "\n$" + salary + " per year");
    }
}
```

Έτσι καλούμε την toString της βασικής κλάσης
Πιο καλή υλοποίηση, μπορεί να έχει φωλιασμένες κλήσεις από προγονικές κλάσεις

super

- Το keyword super χρησιμοποιείται σαν αντικείμενο κλήσης για να καλέσουμε μια μέθοδο της γονικής κλάσης την οποία έχουμε κάνει override.
  - Π.χ., super.toString() για να καλέσουμε την toString της Employee.

- Αν θέλουμε να το ξεχωρίσουμε από την κλήση της toString της SalariedEmployee, μπορούμε να χρησιμοποιήσουμε το this. Μέσα στην SalariedEmployee:
  - super.toString() καλεί την toString της Employee
  - this.toString() καλεί την toString της SalariedEmployee

- Προσοχή: Δεν μπορούμε να έχουμε αλυσιδωτές κλήσεις του super.
  - super.super.toString() είναι λάθος!

Παράδειγμα

```
public class InheritanceDemo
{
    public static void main(String[] args)
    {
       SalariedEmployee sam = new SalariedEmployee("Sam",
                                                   100, 100000);
       HourlyEmployee han = new HourlyEmployee("Han",
                                                  200, 50.5, 40);
       Employee eve = new Employee("Eve",300);

       System.out.println(eve);

       System.out.println(sam);

       System.out.println(han);
       }
}
```

Καλεί τη μέθοδο της Employee

Καλεί τη μέθοδο της SalariedEmployee

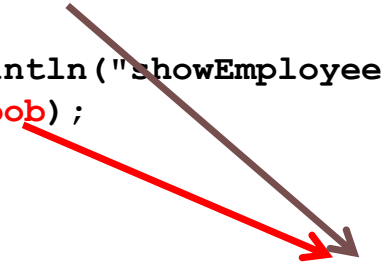
Καλεί τη μέθοδο της HourlyEmployee

Πολλαπλοί τύποι

- Ένα αντικείμενο της παράγωγης κλάσης έχει και τον τύπο της βασικής κλάσης
  - Ένας HourlyEmployee είναι και Employee
  - Υπάρχει μία is-a σχέση μεταξύ των κλάσεων.

- Αυτό μπορούμε να το εκμεταλλευτούμε χρησιμοποιώντας την βασική κλάση όταν θέλουμε να χρησιμοποιήσουμε κάποια από τις παράγωγες.

```
public class IsADemo
{
    public static void main(String[] args)
    {
        HourlyEmployee alice = new HourlyEmployee("Alice",
                                                  100, 50.5, 40);
        SalariedEmployee bob = new SalariedEmployee("Bob",
                                                    200, 100000);

        System.out.println("showEmployee(alice):");
        showEmployee(alice);

        System.out.println("showEmployee(bob):");
        showEmployee(bob);

    }

    public static void showEmployee(Employee employeeObject)
    {
            System.out.println(employeeObject.getName( ));
            System.out.println(employeeObject.getAFM( ));
    }
}
```



Μπορούμε να καλέσουμε τη μέθοδο και με HourlyEmployee και με SalariedEmployee γιατί και οι δύο είναι και Employee.

```
public class Employee
{
        private String name;
        private int AFM;

        public Employee(Employee other){
                this.name = other.name;
                this.AFM = other.AFM;
        }
}
```

```
public class SalariedEmployee extends Employee
{
        public SalariedEmployee(SalariedEmployee other){
                super(other);
                this.salay = other.salary;        }
}
```

Η κλήση του copy constructor της Employee (μέσω της super(other)) γίνεται με ένα αντικείμενο τύπου SalariedEmployee. Αυτό γίνεται γιατί SalariedEmployee is a Employee και το αντικείμενο other έχει και τους δύο τύπους.