Προχωρημένος Προγραμματισμός

Κληρονομικότητα

ΕΛΕΥΘΕΡΙΟΣ ΚΟΣΜΑΣ

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

Περίληψη

Σήμερα ...

- θα συζητήσουμε την ιδέα της κληρονομικότητας
- θα δούμε πώς ορίζουμε την παράγωγη κλάση μιας βασικής κλάσης
- θα μιλήσουμε για ιεραρχία κλάσεων

- θα συζητήσουμε τον τρόπο ορισμού constructor
- θα συζητήσουμε τη χρήση της ενθυλάκωσης στην κληρονομικότητα και τη χρήση της δεσμευμένης λέξης protected

- θα παρουσιάσουμε την έννοια και τη χρησιμότητα της τεχνικής της υπέρβασης μεθόδων, καθώς και τις έννοιες downcasting και upcasting

- θα συζητήσουμε την έννοια του πολυμορφισμού (ή καθυστερημένη δέσμευση)

Λογισμικό για τμήμα ΤΕΙ

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

- ❖ παρομοίως μπορούμε να έχουμε πολλούς διαφορετικούς τύπους φοιτητών ανάλογα με το έτος τους ή τις απαιτήσεις του μαθήματος

Λογισμικό για τμήμα ΤΕΙ ΙΙ

- ? δε 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
  - δε 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

κεντρική έννοια στον αντικειμενοστραφή προγραμματισμό

η ιδέα είναι:

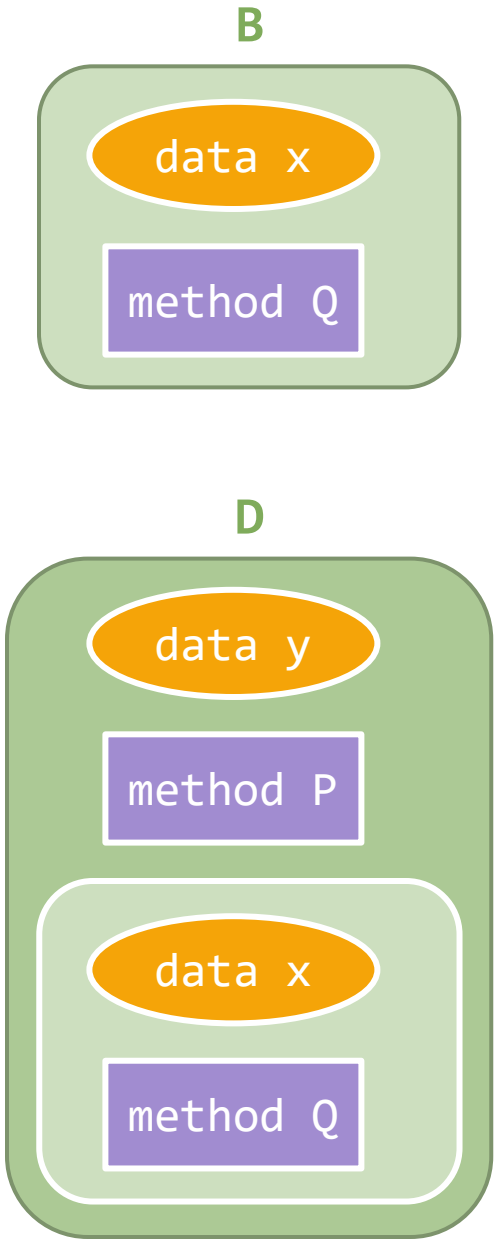
1. να ορίσουμε μια γενική κλάση που έχει κάποια χαρακτηριστικά (πεδία και μεθόδους) που θέλουμε
2. να ορίσουμε εξειδικευμένες παραλλαγές της κλάσης αυτής στις οποίες προσθέτουμε ειδικότερα χαρακτηριστικά

✍ οι εξειδικευμένες κλάσεις λέμε ότι κληρονομούν τα χαρακτηριστικά της γενικής κλάσης

Κληρονομικότητα

- έχουμε μια βασική κλάση (base class) B, με κάποια πεδία και μεθόδους
- θέλουμε να δημιουργήσουμε μια νέα κλάση D η οποία να έχει όλα τα χαρακτηριστικά της B, 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 B και στην οποία προσθέτουμε τα νέα πεδία και μεθόδους

✍ η διαδικασία αυτή λέγετε 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

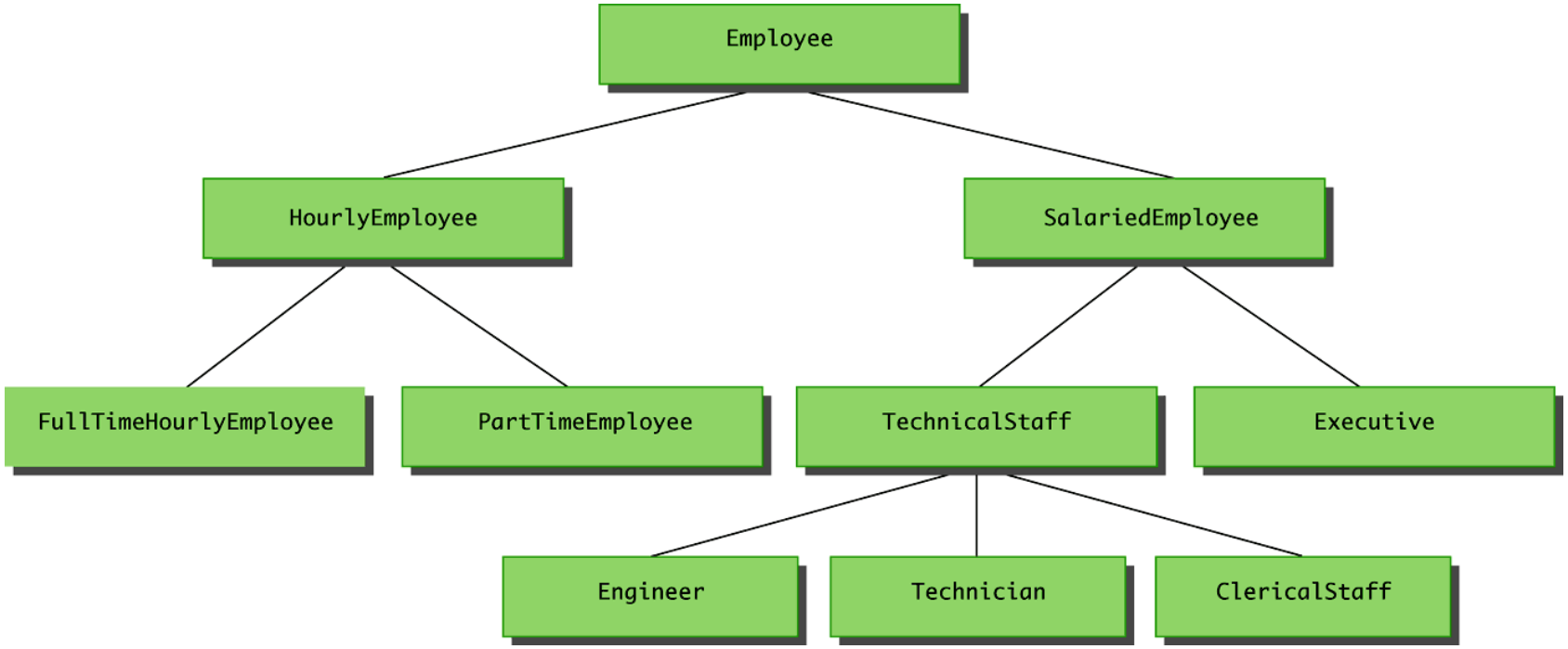
Κληρονομικότητα
Ιεραρχία κλάσεων - Παράδειγμα

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.…

Κληρονομικότητα

Ιεραρχία κλάσεων - Παράδειγμα - Σχηματική απεικόνιση

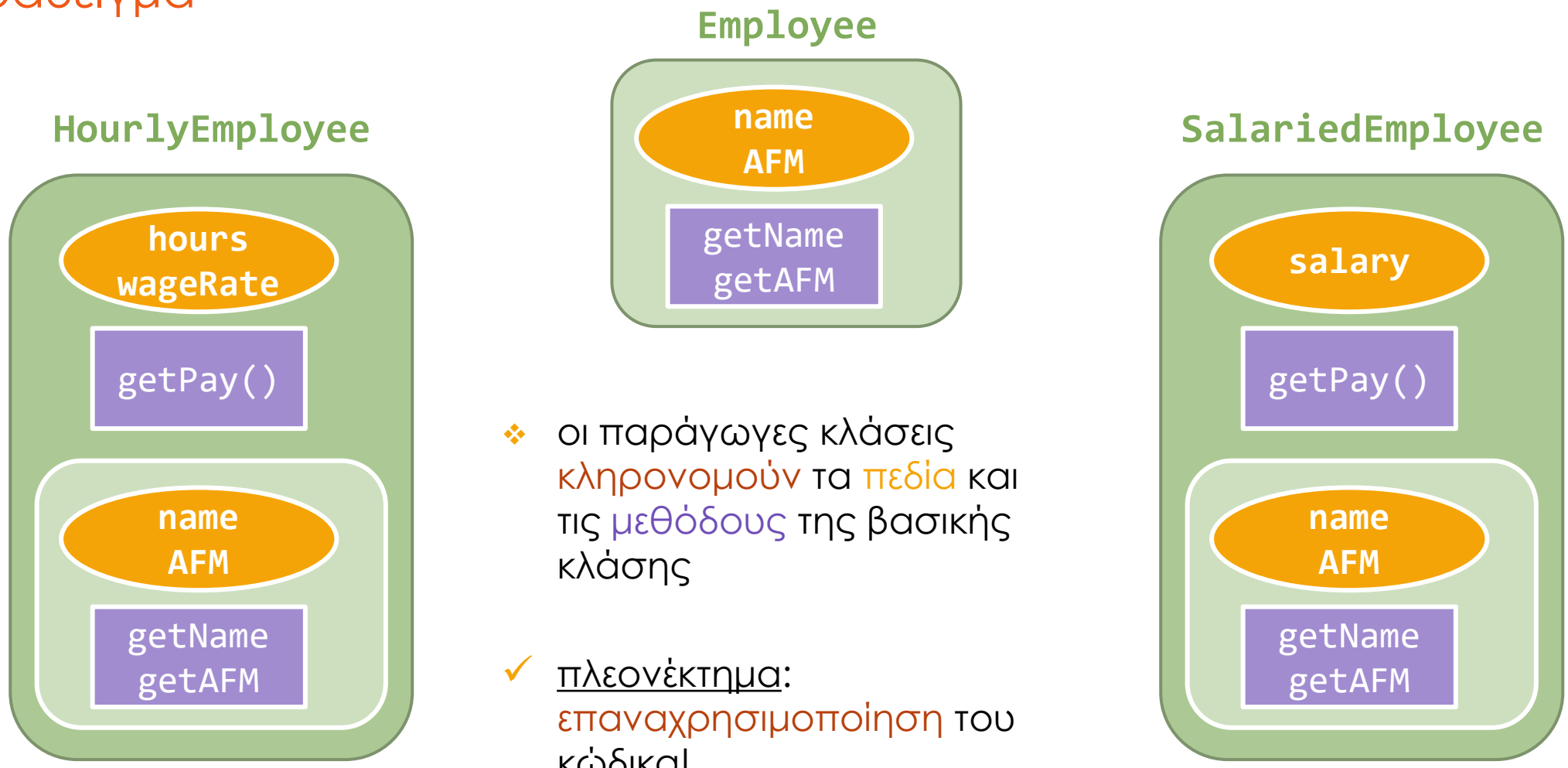


Κληρονομικότητα
Ιεραρχία κλάσεων - Σύνοψη

- 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`

Κληρονομικότητα
Παράδειγμα



- οι παράγωγες κλάσεις κληρονομούν τα πεδία και τις μεθόδους της βασικής κλάσης

✓ πλεονέκτημα: επαναχρησιμοποίηση του κώδικα!

Κληρονομικότητα
Ορολογία

- η βασική κλάση συχνά λέγεται και υπέρ-κλάση (superclass) και η παραγόμενη κλάση υπό-κλάση (subclass)

- 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

Κληρονομικότητα
Συντακτικό - Παράδειγμα

- έστω ότι έχουμε τη 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
```

Κληρονομικότητα

Παράδειγμα - Η βασική κλάση

```
1.  public class Employee {
2.     private String name;
3.     private int AFM;
4.
5.     public Employee() {...}
6.     public Employee(String theName, int theAFM) {...}
7.     public Employee(Employee originalObject) {...}
8.
9.     public String getName() {...}
10.    public void setName(String newName) {...}
11.
12.    public int getAFM() {...}
13.    public void setAFM(int newAFM) {...}
14.
15.    public String toString() {...}
16. }
```

Κληρονομικότητα

Παράδειγμα - Η παράγωγη κλάση HourlyEmployee

```
1.  public class HourlyEmployee extends Employee{
2.     private double wageRate;
3.     private double hours;              // for the month
4.
5.     public HourlyEmployee() {...}
6.     public HourlyEmployee(String theName, int theAFM, double theWageRate, double theHours) {...}
7.     public HourlyEmployee(HourlyEmployee originalObject) {...}
8.
9.     public double getRate() {...}
10.    public void setRate(double newWageRate) {...}
11.
12.    public int getHours() {...}
13.    public void setHours(int hoursWorked) {...}
14.
15.    public double getPay() {
16.       return wageRate * hours;
17.    }
18.
19.    public String toString() {...}
20. }
```

Νέα πεδία για την HourlyEmployee

υπολογίζει το μηνιαίο μισθό

Κληρονομικότητα

Παράδειγμα - Η παράγωγη κλάση SalariedEmployee

```
1.  public class SalariedEmployee extends Employee{
2.    private double salary; // annual
3.
4.    public SalariedEmployee() {...}
5.    public SalariedEmployee(String theName, int theAFM, double theSalary) {...}
6.    public SalariedEmployee(SalariedEmployee originalObject) {...}
7.
8.    public double getSalary() {...}
9.    public void setSalary(double newSalary) {...}
10.
11.   public double getPay() {
12.     return salary/12;
13.   }
14.
15.   public String toString() {...}
16. }
```

νέο πεδίο για την SalariedEmployee

υπολογίζει το μηνιαίο μισθό

είναι διαφορετική από την προηγούμενη!

Κληρονομικότητα

Παράδειγμα - Η μέθοδος main

```
1.  public class Example1 {
2.    public static void main(String args[]) {
3.      HourlyEmployee alice = new HourlyEmployee("Alice", 100, 50.5, 40);
4.
5.      SalariedEmployee bob = new SalariedEmployee("Bob", 200, 100000);
6.
7.      System.out.println("Alice: "
8.                         + alice.getName() + " "
9.                         + alice.getAFM() + " "
10.                        + alice.getPay());
11.
12.     System.out.println("Bob: "
13.                        + bob.getName() + " "
14.                        + bob.getAFM() + " "
15.                        + bob.getPay());
16.   }
17. }
```

μέθοδοι της βασικής κλάσης (Employee)

μέθοδοι των παράγωγων κλάσεων

Constructors

Κληρονομικότητα

Παράδειγμα - Η βασική κλάση - Constructor

```
1.  public class Employee {
2.    private String name;
3.    private int AFM;
4.
5.    public Employee() {
6.      name = "no name";
7.      AFM = 0;
8.    }
9.
10.   public Employee(String theName, int theAFM) {
11.     if (theName == null || theAFM <= 0) {
12.        System.out.println("Fatal Error creating employee.");
13.        System.exit(0);
14.     }
15.     name = theName;
16.     AFM = theAFM;
17.   }
18. }
```

Κληρονομικότητα

Παράδειγμα - Η παράγωγη κλάση HourlyEmployee - Constructor

```
1.  public class HourlyEmployee extends Employee {
2.    private double wageRate;
3.    private double hours;          // for the month
4.
5.    public HourlyEmployee(String theName, int theAFM, double theWageRate, double theHours)
6.    {
7.      super(theName, theAFM);
8.      if (theWageRate >= 0 && theHours >= 0) {
9.         wageRate = theWageRate;
10.        hours = theHours;
11.     } else {
12.        System.out.println("Fatal Error: creating an illegal hourly employee.");
13.        System.exit(0);
14.     }
15.   }
16. }
```

- με τη λέξη κλειδί super αναφερόμαστε στη 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

Κληρονομικότητα

Παράδειγμα - Η παράγωγη κλάση SalariedEmployee - Constructor

```
1.  public class SalariedEmployee extends Employee {
2.    private double salary; // annual
3.
4.    public SalariedEmployee(String theName, int theAFM, double theSalary) {
5.      super(theName, theAFM);
6.      if (theSalary >= 0) {
7.         salary = theSalary;
8.      } else {
9.         System.out.println("Fatal Error: Negative salary.");
10.        System.exit(0);
11.     }
12.   }
13. }
```

Κληρονομικότητα

Παράδειγμα - Η παράγωγη κλάση SalariedEmployee - Constructor II

```
1.  public class SalariedEmployee extends Employee {
2.    private double salary; // annual
3.
4.    public SalariedEmployee() {
5.      super();
6.      salary = 0;
7.    }
8.  }
```

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

Κληρονομικότητα

Παράδειγμα - Η παράγωγη κλάση SalariedEmployee - Constructor III

```
1.  public class SalariedEmployee extends Employee {
2.    private double salary; // annual
3.
4.    public SalariedEmployee(String theName, int theAFM) {
5.      salary = 0;
6.    }
7.  }
```

- ? πώς θα αρχικοποιηθεί το αντικείμενο στην περίπτωση που κληθεί αυτός ο constructor?
  - εφόσον δεν καλείται κάποιος constructor της γονικής κλάσης → θα κληθεί ο default constructor
    - θα αρχικοποιήσει το όνομα στο "no name" και το ΑΦΜ στο μηδέν

Κληρονομικότητα

Παράδειγμα - Η παράγωγη κλάση SalariedEmployee - Constructor IV

```
1.  public class SalariedEmployee extends Employee {
2.    private double salary; // annual
3.
4.    public SalariedEmployee(String theName, int theAFM) {
5.      super(theName, theAFM);
6.      salary = 0;
7.    }
8.  }
```

- αν θέλουμε να αρχικοποιήσουμε το όνομα και το ΑΦΜ (με τις τιμές **theName** και **theAFM**, αντίστοιχα) θα πρέπει να καλέσουμε τον αντίστοιχο constructor της γονικής κλάσης

Κληρονομικότητα

Παράδειγμα - Η παράγωγη κλάση SalariedEmployee - Constructor this

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

```
1.  public class SalariedEmployee extends Employee {
2.     private double salary;    // annual
3.
4.     public SalariedEmployee(String theName, int theAFM, double theSalary) {
5.        super(theName, theAFM);
6.        if (theSalary >= 0) {
7.           salary = theSalary;
8.        } else {
9.           System.out.println("Fatal Error: Negative salary.");
10.          System.exit(0);
11.       }
12.    }
13.
14.    public SalariedEmployee() {
15.       this("no name", 0, 0);
16.    }
17. }
```

Καλεί έναν άλλο constructor της ίδιας κλάσης

Κληρονομικότητα

Παράδειγμα - Η παράγωγη κλάση SalariedEmployee - Constructor V

- ? γιατί να μην κάνουμε κάτι πιο απλό;
  - ☞ απευθείας ανάθεση τιμών στα πεδία της γονικής κλάσης!
- × **ΛΑΘΟΣ!!**
  - ▶ οι παραγόμενες κλάσεις **δεν** έχουν πρόσβαση στα **private** πεδία και τις **private** μεθόδους της βασικής κλάσης

```
1.  public class SalariedEmployee extends Employee {
2.    private double salary; // annual
3.
4.    public SalariedEmployee(String theName, int theAFM, double theSalary) {
5.      name = theName;
6.      AFM = theAFM;
7.      salary = theSalary;
8.    }
9.  }
```

Κληρονομικότητα και Ενθυλάκωση

οι παραγόμενες κλάσεις

1. κληρονομούν την πληροφορία (πεδία, μεθόδους) που έχει και η γονική κλάση
   - ένα αντικείμενο `SalariedEmployee` έχει πληροφορία για το όνομα και το ΑΦΜ του υπαλλήλου
2. δε μπορούν (ούτε) να διαβάσουν/αλλάξουν τα `private` πεδία (ούτε τις `private` μεθόδους της γονικής κλάσης)
   - ✖ στην περίπτωση της `SalariedEmployee`, δε μπορούμε να αλλάξουμε ή να διαβάσουμε το **`name`**
   - ☞ θα πρέπει να χρησιμοποιήσουμε τις `public` μεθόδους **`setName`**, **`getName`**
   - ☞ για τον constructor πρέπει να καλέσουμε τη **`super`**

✍ με αυτό τον τρόπο προστατεύουμε τα δεδομένα της γονικής κλάσης από κώδικα εκτός της κλάσης

Κληρονομικότητα και Ενθυλάκωση
Παράδειγμα

```
1.  public class Employee
2.  {
3.    private void doSomething() {
4.      System.out.println("Do something");
5.    }
6.  }
7.
8.  public class SalariedEmployee extends Employee
9.  {
10.   public void doSomethingMore() {
11.     doSomething();
12.     System.out.println("and more");
13.   }
14. }
```

ΛΑΘΟΣ!

η doSomething είναι private στη γονική κλάση!

Κληρονομικότητα
Protected μέλη

οι παραγόμενες κλάσεις

1. έχουν πρόσβαση σε όλα τα `public` πεδία και μεθόδους της γονικής κλάσης
2. δεν έχουν πρόσβαση στα `private` πεδία και μεθόδους
   - παρά μόνο μέσω `public` μεθόδων `set*` και `get*`

- `protected`: αν κάποια πεδία και μέθοδοι είναι `protected` μπορούν να τα "δουν" όλοι οι απόγονοι της κλάσης
  - δε συνίστατται

- package access: αν δεν προσδιορίσετε `public`, `private`, ή `protected` access τότε η προεπιλεγμένη συμπεριφορά είναι ότι το πεδίο ή η μέθοδος είναι προσβάσιμη από άλλες κλάσεις μέσα στο ίδιο πακέτο

Κληρονομικότητα

Παράδειγμα - Η βασική κλάση - Constructor - Private πεδία

```
1.  public class Employee {
2.     private String name = "default";
3.     private Date hireDate = new Date(18,12,2018);
4.
5.     public Employee() {
6.         System.out.println("empty constructor");
7.     }
8.
9.     public Employee(String theName, String theDate) {
10.       name = theName;
11.       hireDate = new Date(theDate);
12.    }
13. }
14.
15. public class HourlyEmployee extends Employee {
16.    private double wageRate;
17.    private double hours;                    // for the month
18.
19.    public HourlyEmployee(String theName, String theDate, double theWageRate, double theHours) {
20.       name = theName;
21.       hireDate = new Date(theDate);
22.       wageRate = theWageRate;
23.       hours = theHours;
24.    }
25. }
```

ΛΑΘΟΣ!

χτυπάει λάθος η πρόσβαση σε private πεδία

Κληρονομικότητα

Παράδειγμα - Η βασική κλάση - Constructor - Protected πεδία

```
1.  public class Employee {
2.     protected String name = "default";
3.     protected Date hireDate = new Date(18,12,2018);
4.
5.     public Employee() {
6.         System.out.println("empty constructor");
7.     }
8.
9.     public Employee(String theName, String theDate) {
10.       name = theName;
11.       hireDate = new Date(theDate);
12.    }
13. }
14.
15. public class HourlyEmployee extends Employee {
16.    private double wageRate;
17.    private double hours;                  // for the month
18.
19.    public HourlyEmployee(String theName, String theDate, double theWageRate, double theHours) {
20.       name = theName;
21.       hireDate = new Date(theDate);
22.       wageRate = theWageRate;
23.       hours = theHours;
24.    }
25. }
```

σωστή η πρόσβαση σε protected πεδία

Κληρονομικότητα
Πολλαπλοί τύποι

- ένα αντικείμενο της παράγωγης κλάσης έχει και τον τύπο της βασικής κλάσης
  - π.χ. ένα αντικείμενο `HourlyEmployee` είναι και αντικείμενο `Employee`
  - υπάρχει μία is-a σχέση μεταξύ των κλάσεων

- αυτό μπορούμε να το εκμεταλλευτούμε χρησιμοποιώντας τη βασική κλάση όταν θέλουμε να χρησιμοποιήσουμε κάποια από τις παράγωγες

Κληρονομικότητα

Πολλαπλοί τύποι - 1ο Παράδειγμα

```
1.  public class IsADemo {
2.    public static void main(String args[]) {
3.      HourlyEmployee alice = new HourlyEmployee("Alice", 100, 50.5, 40);
4.      SalariedEmployee bob = new SalariedEmployee("Bob", 200, 100000);
5.
6.      System.out.println("showEmployee(alice):");
7.      showEmployee(alice);
8.      System.out.println("showEmployee(bob):");
9.      showEmployee(bob);
10.   }
11.
12.   public static void showEmployee (Employee employeeObject) {
13.     System.out.println(employeeObject.getName());
14.     System.out.println(employeeObject.getAFM());
15.   }
16. }
```

Μπορούμε να καλέσουμε τη μέθοδο με αντικείμενα HourlyEmployee και SalariedEmployee, γιατί και τα δύο είναι και αντικείμενα Employee!

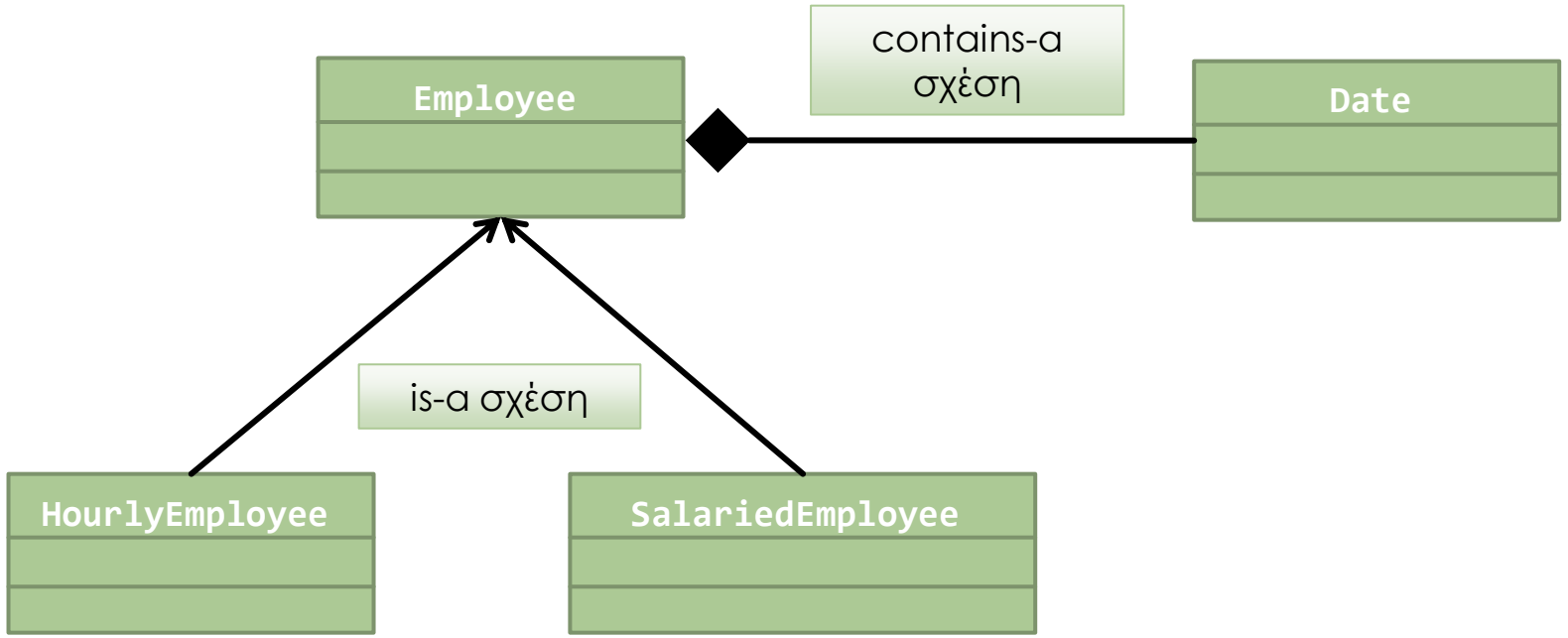
Κληρονομικότητα

Πολλαπλοί τύποι - 2ο Παράδειγμα

```
1.  public class Employee {
2.     private String name;
3.     private int AFM;
4.
5.     public Employee(Employee other) {
6.        this.name = other.name;
7.        this.AFM = other.AFM;
8.     }
9.  }
10.
11. public class SalariedEmployee extends Employee {
12.    private double salary;    // annual
13.
14.    public SalariedEmployee(SalariedEmployee other) {
15.       super(other);
16.       this.salary = other.salary;
17.    }
18. }
```

- η κλήση του copy constructor της Employee (μέσω της super(other)) γίνεται με ένα αντικείμενο τύπου SalariedEmployee
- αυτό είναι σωστό γιατί το αντικείμενο SalariedEmployee είναι (is a) και αντικείμενο Employee
  - οπότε, το αντικείμενο other έχει και τους δύο τύπους

Κληρονομικότητα
Παράδειγμα - Αναπαράσταση (UML διάγραμμα)



Υπέρβαση Μεθόδων (Method Overriding)

Κληρονομικότητα
Υπέρβαση μεθόδων (method overriding)

- μία μέθοδος που ορίζεται στην βασική κλάση μπορούμε να την ορίσουμε ξανά στην παράγωγη κλάση με διαφορετικό τρόπο
- παράδειγμα: η μέθοδος `toString()`
  - την ορίζουμε ξανά για κάθε παραγόμενη κλάση ώστε να παράγει το `String` που επιθυμούμε

☞ αυτό λέγετε υπέρβαση της μεθόδου (method overriding)

✍ η υπέρβαση μεθόδων διαφέρει από την υπερφόρτωση!

- στην υπερφόρτωση αλλάζουμε την υπογραφή της μεθόδου
- στην υπέρβαση έχουμε την ίδια υπογραφή, απλά αλλάζει το σημείο ορισμού της (γίνεται στην παραγόμενη κλάση)

Κληρονομικότητα

Υπέρβαση μεθόδων (method overriding) - Παράδειγμα - Employee

```
1.  public class Employee {
2.    private String name;
3.    private int AFM;
4.
5.    public Employee() {...}
6.    public Employee(String theName, int theAFM) {...}
7.    public Employee(Employee originalObject) {...}
8.
9.    public String getName() {...}
10.   public void setName(String newName) {...}
11.
12.   public int getAFM() {...}
13.   public void setAFM(int newAFM) {...}
14.
15.   public String toString() {
16.     return (name + " " + AFM);
17.   }
18. }
```

ορισμός toString στη βασική κλάση

Κληρονομικότητα

Υπέρβαση μεθόδων (method overriding) - Παράδειγμα - HourlyEmployee

```
1.  public class HourlyEmployee extends Employee{
2.     private double wageRate;
3.     private double hours;              // for the month
4.
5.     public HourlyEmployee() {...}
6.     public HourlyEmployee(String theName, int theAFM, double theWageRate, double theHours) {...}
7.     public HourlyEmployee(HourlyEmployee originalObject) {...}
8.
9.     public double getRate() {...}
10.    public void setRate(double newWageRate) {...}
11.
12.    public int getHours() {...}
13.    public void setHours(int hoursWorked) {...}
14.
15.    public double getPay() { return wageRate * hours; }
16.
17.    public String toString() {
18.        return (getName() + " " + getAFM() + "\n$" + wageRate + " per hour for " + hours + " hours");
19.    }
20. }
```

ορισμός ξανά της toString στην παραγόμενη κλάση

Κληρονομικότητα

Υπέρβαση μεθόδων (method overriding) - Παράδειγμα - SalariedEmployee

```
1.  public class SalariedEmployee extends Employee{
2.    private double salary; // annual
3.
4.    public SalariedEmployee() {...}
5.    public SalariedEmployee(String theName, int theAFM, double theSalary) {...}
6.    public SalariedEmployee(SalariedEmployee originalObject) {...}
7.
8.    public double getSalary() {...}
9.    public void setSalary(double newSalary) {...}
10.
11.   public double getPay() { return salary/12; }
12.
13.   public String toString() {
14.       return (getName() + " " + getAFM() + "\n$" + salary + " per year");
15.   }
16. }
```

ορισμός ξανά της toString στην παραγόμενη κλάση

Κληρονομικότητα

Υπέρβαση μεθόδων (method overriding) - Παράδειγμα - SalariedEmployee II

```
1.  public class SalariedEmployee extends Employee{
2.    private double salary; // annual
3.
4.    public SalariedEmployee() {...}
5.    public SalariedEmployee(String theName, int theAFM, double theSalary) {...}
6.    public SalariedEmployee(SalariedEmployee originalObject) {...}
7.
8.    public double getSalary() {...}
9.    public void setSalary(double newSalary) {...}
10.
11.   public double getPay() { return salary/12; }
12.
13.   public String toString() {
14.       return (super.toString() + "\n$" + salary + " per year");
15.   }
16. }
```

έτσι καλούμε την toString της βασικής κλάσης

πιο καλή υλοποίηση, μπορεί να έχει φωλιασμένες κλήσεις από προγονικές κλάσεις

Κληρονομικότητα
Υπέρβαση μεθόδων - Super

- η δεσμευμένη λέξη `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 καλέσουμε την `toString` της `SalariedEmployee`, μπορούμε να χρησιμοποιήσουμε το `this`

  π.χ. μέσα στην `SalariedEmployee`:
  - `super.toString()` καλεί την `toString` της `Employee`
  - `this.toString()` καλεί την `toString` της `SalariedEmployee`

✍ προσοχή: δε μπορούμε να έχουμε αλυσιδωτές κλήσεις του `super`

- ✘ `super.super.toString()` → είναι λάθος!

Κληρονομικότητα

Υπέρβαση μεθόδων (method overriding) - Παράδειγμα - main

```
1.  public class InheritanceDemo {
2.    public static void main(String args[]) {
3.      HourlyEmployee alice = new HourlyEmployee("Alice", 100, 50.5, 40);
4.      SalariedEmployee bob = new SalariedEmployee("Bob", 200, 100000);
5.      Employee eve = new Employee("Eve", 300);
6.
7.      System.out.println(alice);
8.
9.      System.out.println(bob);
10.
11.     System.out.println(eve);
12.   }
13. }
```

καλεί τη μέθοδο toString της HourlyEmployee

καλεί τη μέθοδο toString της SalariedEmployee

καλεί τη μέθοδο toString της Employee

Κληρονομικότητα

Υπέρβαση μεθόδων (method overriding) - Αλλαγή επιστρεφόμενου τύπου

- μια αλλαγή που μπορούμε να κάνουμε στην υπογραφή της κλάσης που υπερβαίνουμε είναι να αλλάξουμε τον επιστρεφόμενο τύπο σε αυτόν μιας παράγωγης κλάσης
  - ουσιαστικά δεν είναι αλλαγή αφού η παράγωγη κλάση έχει και τον τύπο της γονικής κλάσης

Κληρονομικότητα

Υπέρβαση μεθόδων (method overriding) - Αλλαγή επιστρεφόμενου τύπου

```
1.  public class Employee {
2.    private String name = "default";
3.    private Date hireDate;
4.
5.    public Employee CreateCopy()
6.    {
7.      return new Employee(this);
8.    }
9.  }
```

Κληρονομικότητα

Υπέρβαση μεθόδων (method overriding) - Αλλαγή επιστρεφόμενου τύπου

```
1.  public class HourlyEmployee extends Employee {
2.    private double wageRate;
3.    private double hours;              // for the month
4.
5.    public HourlyEmployee CreateCopy() {
6.      return new HourlyEmployee(this);
7.    }
8.  }
```

ο επιστρεφόμενος τύπος αλλάζει από Employee σε HourlyEmployee στην υπέρβαση

ουσιαστικά όμως δεν υπάρχει αλλαγή μιας και κάθε αντικείμενο HourlyEmployee είναι και Employee

Κληρονομικότητα

Υπέρβαση μεθόδων (method overriding) - Αλλαγή επιστρεφόμενου τύπου

```
1.  public class SalariesEmployee extends Employee {
2.    private double salary;              // annual
3.
4.    public SalariedEmployee CreateCopy() {
5.      return new SalariedEmployee(this);
6.    }
7.  }
```

ο επιστρεφόμενος τύπος αλλάζει από Employee σε SalariedEmployee στην υπέρβαση

ουσιαστικά όμως δεν υπάρχει αλλαγή μιας και κάθε αντικείμενο SalariedEmployee είναι και Employee

Κληρονομικότητα

Υπέρβαση μεθόδων (method overriding) - toString και equals

όπως έχουμε ήδη αναφέρει

- η Java για κάθε αντικείμενο «περιμένει» να δει τις μεθόδους `toString` και `equals`
  - αυτό σημαίνει ότι
    - οι μέθοδοι αυτές ορίζονται στην κλάση `Object` που είναι ο πρόγονος όλων των κλάσεων και
    - κάθε νέα κλάση μπορεί να τις υπερβεί (override)
  - είδαμε παραδείγματα για τον τρόπο υπέρβασης της μεθόδου `toString`

Κληρονομικότητα

Υπέρβαση μεθόδων (method overriding) - equals

- η `equals` στην κλάση `Object` ορίζεται ως:

```
public boolean equals(Object other)
```

- αν στην κλάση `Employee` την ορίσουμε ως:

```
public boolean equals(Employee other)
```

- τότε, αλλάζουμε την υπογραφή της κλάσης → άρα δεν κάνουμε υπέρβαση, αλλά υπερφόρτωση της `equals`

? πώς θα την ορίσουμε ώστε να κάνουμε υπέρβαση;

Κληρονομικότητα

Υπέρβαση μεθόδων (method overriding) - Υπέρβαση της equals

```
1.  public class Employee {
2.    private String name;
3.    private Date hireDate;
4.
5.    public boolean equals(Object otherObject) {
6.      if (otherObject == null)
7.         return false;
8.      else if (getClass() != otherObject.getClass( ))
9.         return false;
10.     else {
11.        Employee otherEmployee = (Employee)otherObject;
12.        return (name.equals(otherEmployee.name)
13.                && hireDate.equals(otherEmployee.hireDate));
14.     }
15.   }
16. }
```

getClass: μέθοδος της Object, επιστρέφει μια αναπαράσταση της κλάσης του αντικειμένου

downcasting: μετατροπή ενός αντικειμένου από μια υψηλότερη σε μία χαμηλότερη κλάση

Το downcasting δεν είναι πάντα δυνατόν και αν δε γίνει σωστά μπορεί να προκαλέσει λάθη κατά την εκτέλεση του προγράμματος

Downcasting

Κληρονομικότητα
Downcasting

```
1.  public class DowncastingExample {
2.    public static void main(String args[]) {
3.      SalariedEmployee sam = new SalariedEmployee("Sam", 200, 100000);
4.      Employee eve = new Employee("Eve", 300);
5.
6.      SalariedEmployee eve2 = eve;
7.      if (sam.getName().equals(eve.getName())) {
8.         System.out.println("Same name");
9.      } else {
10.        System.out.println("Different name");
11.     }
12.   }
13. }
```

- στην περίπτωση αυτή προσπαθούμε να κάνουμε το downcasting έμμεσα, αναθέτοντας μια μεταβλητή Employee σε μια μεταβλητή SalariedEmployee
- θα μας χτυπήσει **λάθος** κατά τη μεταγλώτιση

Κληρονομικότητα
Downcasting II

```
1.  public class DowncastingExample {
2.    public static void main(String args[]) {
3.      SalariedEmployee sam = new SalariedEmployee("Sam", 200, 100000);
4.      Employee eve = new Employee("Eve", 300);
5.
6.      SalariedEmployee eve2 = (SalariedEmployee) eve;
7.      if (sam.getName().equals(eve2.getName())) {
8.         System.out.println("Same name");
9.      } else {
10.        System.out.println("Different name");
11.     }
12.   }
13. }
```

- στην περίπτωση αυτή θα μας χτυπήσει **λάθος** στο τρέξιμο παρότι χρησιμοποιούμε μόνο την κοινή μέθοδο getName()
- το πρόγραμμα προβλέπει ότι μπορεί να υπάρχει πρόβλημα
- δε γίνεται να μετατρέψουμε έναν Employee σε SalariedEmployee(καθώς ο Employee δεν έχει όλα τα πεδία που χρειάζεται ένας SalariedEmployee)

Κληρονομικότητα
Downcasting III

```
1.  public class DowncastingExample {
2.    public static void main(String args[]) {
3.      SalariedEmployee sam = new SalariedEmployee("Sam", 200, 100000);
4.      Employee eve = new Employee("Eve", 300);
5.      method(sam,sam);
6.    }
7.
8.    private static void method(SalariedEmployee sEmp, Employee emp) {
9.      SalariedEmployee sEmp2 = (SalariedEmployee) emp;
10.     if (sEmp.getName().equals(sEmp2.getName())) {
11.        System.out.println("Same name");
12.     } else {
13.        System.out.println("Different name");
14.     }
15.   }
16. }
```

στην περίπτωση αυτή το downcasting δε χτυπάει λάθος γιατί **μπορούμε** να καλέσουμε σωστά τη method() με SalariedEmployee αντικείμενο

Κληρονομικότητα
Downcasting IV

```
1.  public class DowncastingExample {
2.    public static void main(String args[]) {
3.      SalariedEmployee sam = new SalariedEmployee("Sam", 200, 100000);
4.      Employee eve = new Employee("Eve", 300);
5.      method(sam,eve);
6.    }
7.
8.    private static void method(SalariedEmployee sEmp, Employee emp) {
9.      SalariedEmployee sEmp2 = (SalariedEmployee) emp;
10.     if (sEmp.getName().equals(sEmp2.getName())) {
11.        System.out.println("Same name");
12.     } else {
13.        System.out.println("Different name");
14.     }
15.   }
16. }
```

αν όμως καλέσουμε τη method() με αντικείμενο Employee θα πάρουμε **λάθος**

Κληρονομικότητα

Downcasting - Σε τι μας χρειάζεται;

```
1.  import java.util.Random;
2.
3.  public class DowncastingExample {
4.     public static void main(String args[]) {
5.        SalariedEmployee[] sEmployees = new SalariedEmployee[4];
6.        sEmployees[0] = new SalariedEmployee("Employee 10", 10, 1000);
7.        sEmployees[1] = new SalariedEmployee("Employee 11", 11, 2000);
8.        sEmployees[2] = new SalariedEmployee("Employee 12", 12, 3000);
9.        sEmployees[3] = new SalariedEmployee("Employee 13", 13, 4000);
10.
11.       SalariedEmployee rand = (SalariedEmployee) randomSelection(sEmployees);
12.       System.out.println(rand);
13.       System.out.println("Salary per month " + rand.getPay());
14.    }
15.
16.    private static Employee randomSelection(Employee[] employees) {
17.       Random rdnGen = new Random();
18.       int r = rdnGen.nextInt(employees.length);
19.       return employees[r];
20.    }
21. }
```

- έχουμε μια γενική μέθοδο **randomSelection** που επιλέγει ένα τυχαίο στοιχείο από ένα πίνακα με **Employee**
- θέλουμε να την χρησιμοποιήσουμε σε ένα πίνακα με **SalariedEmployee**
- θέλουμε να καλέσουμε τη μέθοδο **getPay** για τυπώσουμε το μηνιαίο μισθό
- χρειαζόμαστε **downcasting**

Κληρονομικότητα
Upcasting

- η ανάθεση στην αντίθετη κατεύθυνση (upcasting) μπορεί να γίνει χωρίς να χρειάζεται casting (μετατροπή τύπου)
  - μπορούμε να κάνουμε μια ανάθεση `x = y` δύο αντικειμένων αν:
    - τα δύο αντικείμενα να είναι της ίδιας κλάσης ή
    - η κλάση του αντικειμένου που ανατίθεται (`y`) είναι απόγονος της κλάσης του αντικειμένου στο οποίο γίνεται η ανάθεση (`x`)
  - π.χ. ο παρακάτω κώδικας δουλεύει χωρίς πρόβλημα:

```
Employee emp;
HourlyEmployee hEmp = new HourlyEmployee();
emp = hEmp;
```

Κληρονομικότητα

Upcasting - Παράδειγμα

```
1.  public class IsADemo {
2.    public static void main(String args[]) {
3.      HourlyEmployee alice = new HourlyEmployee("Alice", 100, 50.5, 40);
4.      SalariedEmployee bob = new SalariedEmployee("Bob", 200, 100000);
5.
6.      System.out.println("showEmployee(alice):");
7.      showEmployee(alice);
8.      System.out.println("showEmployee(bob):");
9.      showEmployee(bob);
10.   }
11.
12.   public static void showEmployee (Employee employeeObject) {
13.     System.out.println(employeeObject.getName());
14.     System.out.println(employeeObject.getAFM());
15.   }
16. }
```

Όταν καλούμε τη **showEmployee()** έμμεσα κάνουμε τις αναθέσεις:

- `employeeObject = alice;`
- `employeeObject = bob;`

Πολυμορφισμός (Polymorphism)
ή
Καθυστερημένη Δέσμευση (Late Binding)

Πολυμορφισμός ή Καθυστερημένη Δέσμευση

1ο Παράδειγμα - Ερωτήματα

```
1.  public class IsADemo {
2.     public static void main(String args[]) {
3.        HourlyEmployee alice = new HourlyEmployee("Alice", 100, 50.5, 40);
4.        SalariedEmployee bob = new SalariedEmployee("Bob", 200, 100000);
5.
6.        System.out.println("showEmployee(alice):");
7.        showEmployee(alice);
8.        System.out.println("showEmployee(bob):");
9.        showEmployee(bob);
10.    }
11.
12.    public static void showEmployee (Employee employeeObject) {
13.       System.out.println(employeeObject);
14.    }
15. }
```

- ? τι θα τυπώσει η `showEmployee` όταν την καλέσουμε με ορίσματα `alice` και `bob`;
- ? ποια μέθοδος `toString` θα κληθεί?

Πολυμορφισμός ή Καθυστερημένη Δέσμευση

1ο Παράδειγμα - Απάντηση

```
1.  public class IsADemo {
2.     public static void main(String args[]) {
3.        HourlyEmployee alice = new HourlyEmployee("Alice", 100, 50.5, 40);
4.        SalariedEmployee bob = new SalariedEmployee("Bob", 200, 100000);
5.
6.        System.out.println("showEmployee(alice):");
7.        showEmployee(alice);
8.        System.out.println("showEmployee(bob):");
9.        showEmployee(bob);
10.    }
11.
12.    public static void showEmployee (Employee employeeObject) {
13.       System.out.println(employeeObject);
14.    }
15. }
```

Θα καλέσει την **toString** της κλάσης του αντικειμένου που περνάμε σαν όρισμα (**HourlyEmployee** ή **SalariedEmployee**) και **όχι** την κλάση που εμφανίζεται στον ορισμό της παραμέτρου (**Employee**)

ο μηχανισμός αυτός ονομάζεται **late binding** (και/ή **πολυμορφισμός**)

Πολυμορφισμός ή Καθυστερημένη Δέσμευση

- η δέσμευση (binding) αναφέρεται στο συσχετισμό μεταξύ της κλήσης μιας μεθόδου και του ορισμού (κώδικα) της μεθόδου
  1. early binding: η δέσμευση γίνεται κατά τη μεταγλώττιση του προγράμματος
     - π.χ. στην περίπτωση αυτή η μέθοδος `toString()` που θα κληθεί θα είναι η μέθοδος της κλάσης `Employee`, καθώς όταν γίνεται η μεταγλώττιση ο μεταγλωττιστής (compiler) βλέπει το όρισμα ως αντικείμενο της κλάσης `Employee`
  2. late binding: η δέσμευση γίνεται κατά την εκτέλεση του προγράμματος
     - το κάθε αντικείμενο έχει πληροφορία για την κλάση του και τον ορισμό (κώδικα) των μεθόδων του
     - π.χ. στην περίπτωση αυτή η μέθοδος `toString()` που θα κληθεί εξαρτάται από την κλάση που περνάμε σαν όρισμα (`Employee`, `HourlyEmployee` ή `SalariedEmployee`)
       - ανάλογα με το αντικείμενο καλείται η ανάλογη μέθοδος

- στη Java εφαρμόζεται ο μηχανισμός του late binding για όλες τις μεθόδους
  - σε αντίθεση με άλλες γλώσσες προγραμματισμού

Πολυμορφισμός ή Καθυστερημένη Δέσμευση
2ο Παράδειγμα

```
1.  public class LateBindingExample {
2.    public static void main(String args[]) {
3.      Employee[] employeeArray = new Employee[3];
4.      employeeArray[0] = new Employee("charlie", 10);
5.      employeeArray[1] = new HourlyEmployee("alice", 20, 50.5, 40);
6.      employeeArray[2] = new SalariedEmployee("bob", 30, 100000);
7.
8.      for (int i = 0; i < 3; i++) {
9.         System.out.println(employeeArray[i]);
10.     }
11.   }
12. }
```

για κάθε στοιχείο του πίνακα **employeeArray** καλείται **διαφορετική** μέθοδος **toString** ανάλογα με το αντικείμενο που τοποθετήσαμε σε εκείνη τη θέση

Πολυμορφισμός ή Καθυστερημένη Δέσμευση

3ο Παράδειγμα - MySale

όπως έχουμε συζητήσει **δε** συνίσταται η χρήση της protected αλλά τη χρησιμοποιούμε για **απλότητα** στο παράδειγμα

```
1.  public class MySale {
2.     protected String name;
3.     protected double price;
4.
5.     public MySale(String theName, double thePrice) {
6.        name = theName;
7.        price = thePrice;
8.     }
9.
10.    public String toString() { return (name + " Price and total cost = $" + price); }
11.
12.    public double bill() { return price; }
13.
14.    public boolean equalDeals(MySale otherSale) {
15.       return (name.equals(otherSale.name) && this.bill() == otherSale.bill());
16.    }
17.
18.    public boolean lessThan(MySale otherSale) {
19.       return (this.bill() < otherSale.bill());
20.    }
21. }
```

Πολυμορφισμός ή Καθυστερημένη Δέσμευση

3ο Παράδειγμα - MyDiscountSale

```
1.  public class MyDiscountSale extends MySale {
2.     private double discount;
3.
4.     public MyDiscountSale(String theName, double thePrice, double theDiscount) {
5.        super(theName, thePrice);
6.        discount = theDiscount;
7.     }
8.
9.     public double bill() {
10.       double fraction = discount/100;
11.       return (1 - fraction) * price;
12.    }
13.
14.    public String toString() {
15.       return (name + " Price = $" + price
16.               + " Discount = " + discount + "%\n"
17.               + " Total cost = $" + bill());
18.    }
19. }
```

υπέρβαση της μεθόδου **bill()**

υπέρβαση της μεθόδου **toString()**

δεν έχουμε υπέρβαση των μεθόδων **equalDeals** και **lessThan**

Πολυμορφισμός ή Καθυστερημένη Δέσμευση

3ο Παράδειγμα - MyLateBindingDemo

```
1.  public class MyLateBindingDemo {
2.     public static void main(String args[]) {
3.        MySale simple = new MySale("floor mat", 10.00);                    // one item at $10.00
4.        MyDiscountSale discount = new MyDiscountSale("floor mat", 11.00, 10);    // one item at $11.00 with a 10% discount
5.        System.out.println(simple);
6.        System.out.println(discount);
7.        if (discount.lessThan(simple))
8.           System.out.println("Discounted item is cheaper.");
9.        else
10.          System.out.println("Discounted item is not cheaper.");
11.       MySale regularPrice = new MySale("cup", 9.90);                     // one item at $9.90
12.       MyDiscountSale specialPrice = new MyDiscountSale("cup", 11.00, 10);      // one item at $11.00 with a 10% discount
13.       System.out.println(regularPrice);
14.       System.out.println(specialPrice);
15.       if (specialPrice.equalDeals(regularPrice))
16.          System.out.println("Deals are equal.");
17.       else
18.          System.out.println("Deals are not equal.");
19.    }
20. }
```

οι **lessThan()** και **equalDeals()** **κληρονομούνται** από τη **MySale**

- με το μηχανισμό του late binding στην κλήση τους ξέρουμε ότι το αντικείμενο που τις καλεί είναι τύπου **MyDiscountSale**
- ξέρουμε λοιπόν ότι όταν εκτελούμε τον κώδικα των **lessThan()** και **equalDeals()**, η μέθοδος **bill()** που θα πρέπει να καλέσουμε είναι αυτή της **MyDiscountSale** ενώ για το **otherSale.bill()** είναι αυτή της **mySale**

Σύνοψη

- Κληρονομικότητα
  - Ορολογία
    - βασική και παράγωγη κλάση
    - πρόγονος και απόγονος
  - Συντακτικό
  - Constructors
    - super, this
  - Ενθυλάκωση
    - public, protected, private
  - Downcasting και Upcasting
- Ιεραρχία κλάσεων
- Υπέρβαση μεθόδων
  - toString, equals
- Καθυστερημένη Δέσμευση (Late Binding) ή Πολυμορφισμός (Polymorphism)