ΤΕΧΝΙΚΕΣ ΑΝΤΙΚΕΙΜΕΝΟΣΤΡΑΦΟΥΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ

Κληρονομικότητα

Παράδειγμα

- Στο προηγούμενο παράδειγμα 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 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.

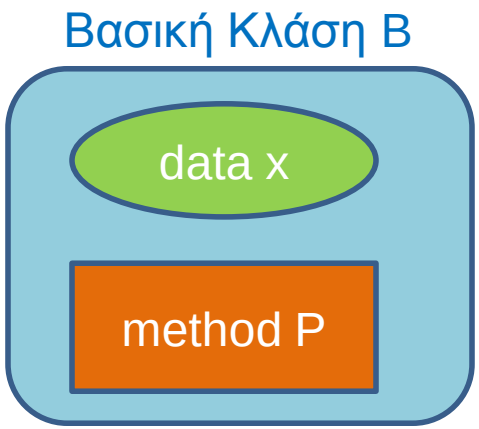
- Αυτό το καταφέρνουμε με την κληρονομικότητα!

Κληρονομικότητα

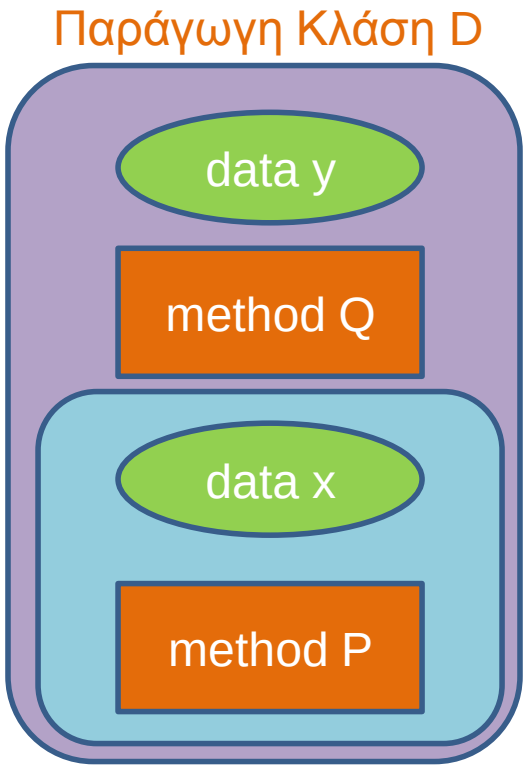
- Η κληρονομικότητα είναι κεντρική έννοια στον αντικειμενοστραφή προγραμματισμό.
- Η ιδέα είναι να ορίσουμε μια γενική κλάση που έχει κάποια χαρακτηριστικά (πεδία και μεθόδους) που θέλουμε και μετά να ορίσουμε εξειδικευμένες παραλλαγές της κλάσης αυτής στις οποίες προσθέτουμε ειδικότερα χαρακτηριστικά.
  - 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 Β και της κλάσης D.
  - Θέλουμε να ορίσουμε πολλαπλές παράγωγες κλάσεις D1, D2, … που η κάθε μία επεκτείνει την Β με διαφορετικό τρόπο.

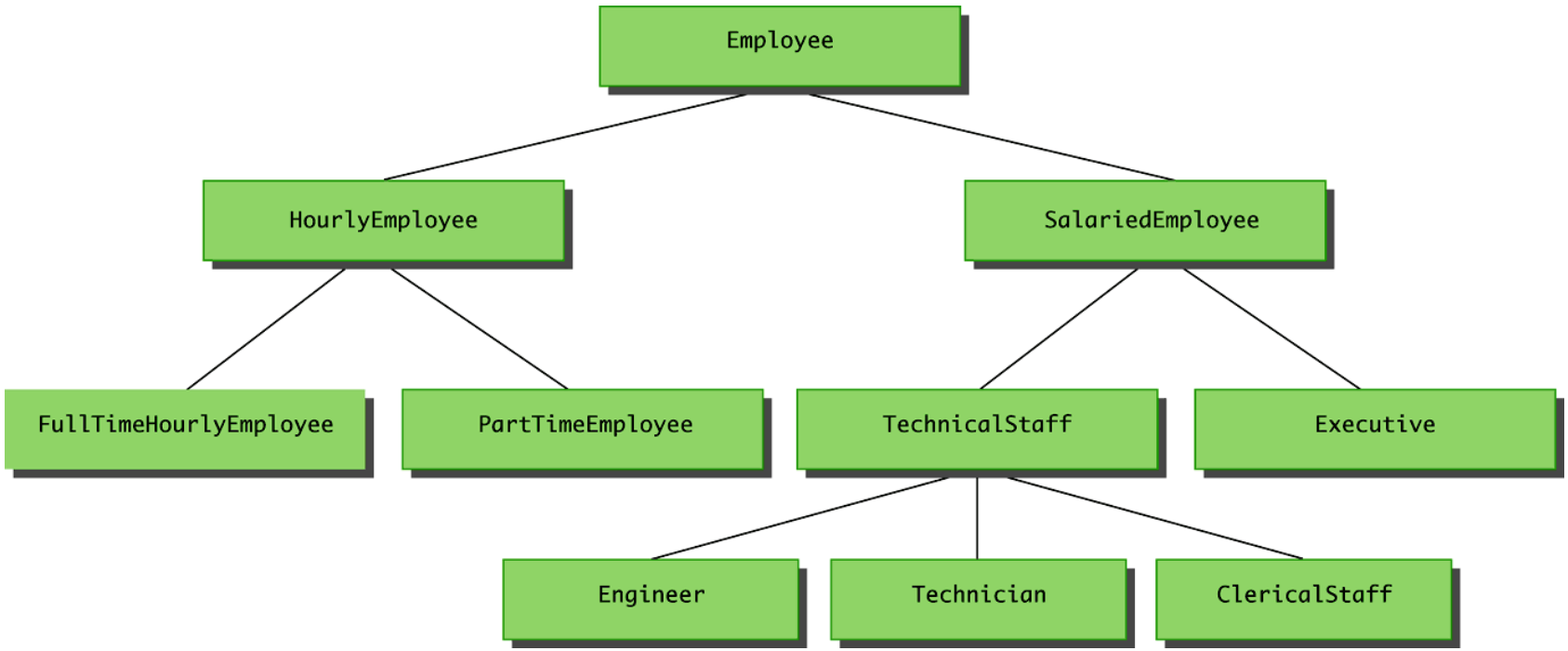
- Μπορούμε να ορίσουμε παράγωγες κλάσεις των παράγωγων κλάσεων.
  - Με αυτό τον τρόπο ορίζεται μια ιεραρχία κλάσεων.

Ιεραρχία κλάσεων (Class Hierarchy)

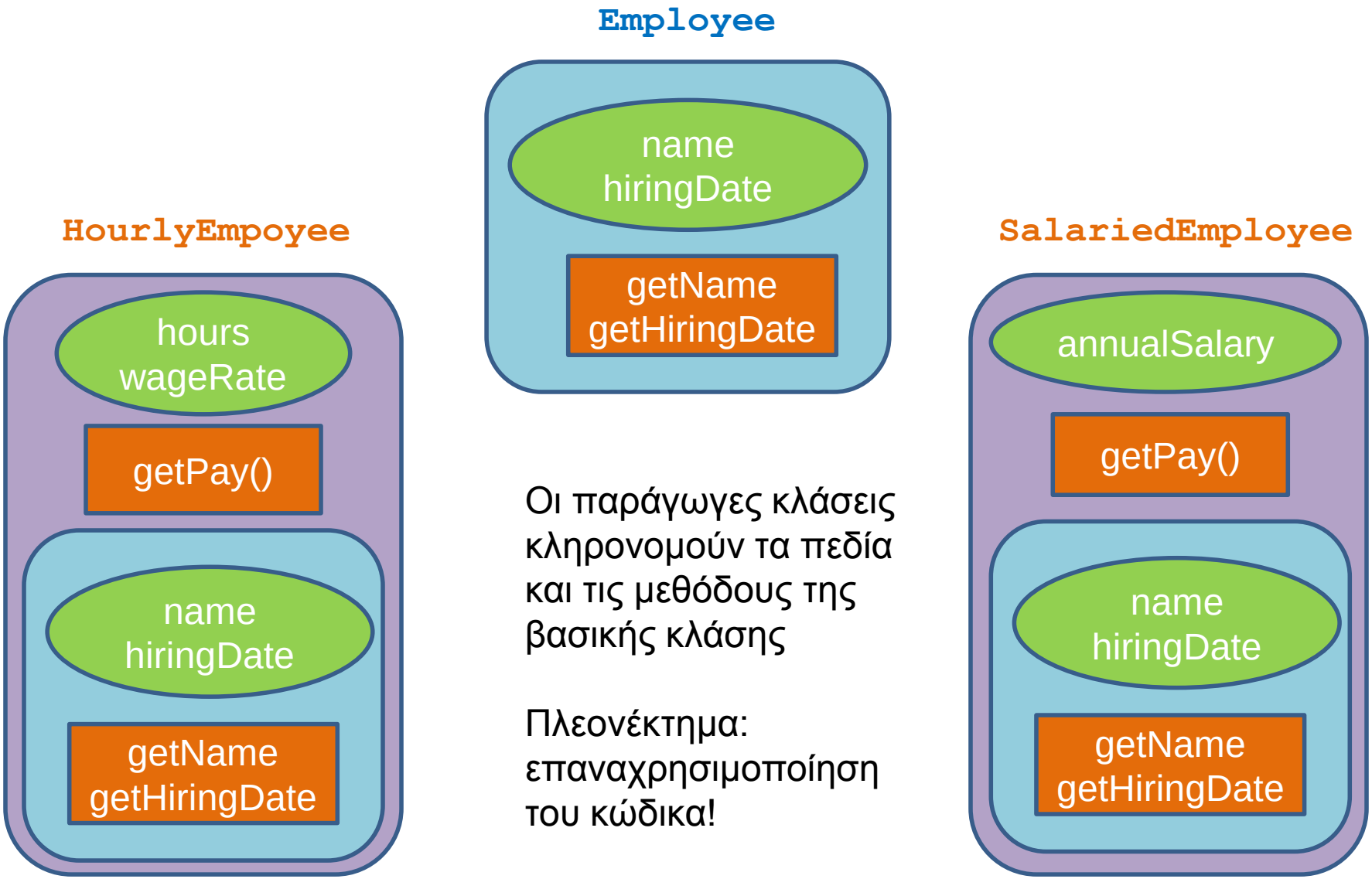
- Παράδειγμα: Έχουμε ένα πρόγραμμα που διαχειρίζεται τους Εργαζόμενους μιας εταιρίας.
  - Όλοι οι εργαζόμενοι έχουν κοινά χαρακτηριστικά το όνομα τους και την ημερομηνία πρόσληψης.
- Οι εργαζόμενοι χωρίζονται σε Ωρομίσθιους και Έμμισθους
  - Διαφορετικά χαρακτηριστικά θα κρατάμε όσον αφορά το μισθό για 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**



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

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

Η βασική κλάση

```
public class Employee
{
    private String name;
    private Date hireDate;

    public Employee( ) { … }

    public Employee(String theName, Date theDate) { … }

    public Employee(Employee originalObject) { … }

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

    public Date getHireDate( ) { … }
    public void setHireDate(Date newDate) { … }

    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, Date theDate,
                          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,
                            Date theDate, 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

Constructor

```
public class Employee
{
    private String name;
    private Date hireDate;


    public Employee(String theName, Date theDate)
    {
        if (theName == null || theDate == null)
        {
            System.out.println("Fatal Error creating employee.");
            System.exit(0);
        }
        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)
    {
        super(theName, theDate);
        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 την ημερομηνία

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

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

    public SalariedEmployee(SalariedEmployee originalObject )
    {
        super(originalObject);
        salary = originalObject.salary;
    }
}
```

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 Date hireDate;

    public Employee( ) { … }

    public Employee(String theName, Date theDate) { … }

    public Employee(Employee originalObject) { … }

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

    public Date getHireDate( ) { … }
    public void setHireDate(Date newDate) { … }

    public String toString()
    {
        return (name + " " + hireDate.toString( ));
    }
}
```

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

    public HourlyEmployee( ) { … }

    public HourlyEmployee(String theName, Date theDate,
                          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( ) + " " + getHireDate( ).toString( )
            + "\n$" + wageRate + " per hour for " + hours + " hours");
    }
}
```

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

    public SalariedEmployee( ) { … }

    public SalariedEmployee(String theName,
                            Date theDate, 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( ) + " " + getHireDate( ).toString( )
                                 + "\n$" + salary + " per year");
    }
}
```

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

    public SalariedEmployee( ) { … }

    public SalariedEmployee(String theName,
                            Date theDate, 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

Παράδειγμα

```
public class InheritanceDemo
{
    public static void main(String[] args)
    {
        HourlyEmployee joe = new HourlyEmployee("Joe Worker",
                             new Date("January", 1, 2004), 50.50, 160);

        System.out.println("joe's longer name is " + joe.getName( ));

        System.out.println("Changing joe's name to Josephine.");
        joe.setName("Josephine");


        System.out.println("joe's record is as follows:");
        System.out.println(joe);
       }
}
```

Καλεί τις μεθόδους της Employee

Καλεί τις μεθόδους 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)
    {
        SalariedEmployee joe = new SalariedEmployee("Josephine",
                                    new Date("January", 1, 2004), 100000);
        HourlyEmployee sam = new HourlyEmployee("Sam",
                               new Date("February", 1, 2003), 50.50, 40);

        System.out.println("joe's longer name is " + joe.getName( ));

        System.out.println("showEmployee(joe) invoked:");
        showEmployee(joe);

        System.out.println("showEmployee(sam) invoked:");
        showEmployee(sam);

    }

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

Μπορούμε να καλέσουμε τη μέθοδο και με HourlyEmployee και με SalariedEmployee

```
public class IsADemo
{
    public static void main(String[] args)
    {
        SalariedEmployee joe = new SalariedEmployee("Josephine",
                                    new Date("January", 1, 2004), 100000);
        HourlyEmployee sam = new HourlyEmployee("Sam",
                               new Date("February", 1, 2003), 50.50, 40);

        System.out.println("joe's longer name is " + joe.getName( ));

        System.out.println("showEmployee(joe) invoked:");
        showEmployee(joe);

        System.out.println("showEmployee(sam) invoked:");
        showEmployee(sam);

    }

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

Θα καλέσει την toString που αντιστοιχεί στο αντικείμενο που περάσαμε ως παράμετρο.