ΤΕΧΝΙΚΕΣ ΑΝΤΙΚΕΙΜΕΝΟΣΤΡΑΦΟΥΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ

Κληρονομικότητα

Η ΜΕΤΑΒΛΗΤΗ THIS

Η μεταβλητή this

- Η μεταβλητή (παράμετρος) this
  - Μια κρυφή παράμετρος η οποία περνάει σε κάθε μέθοδο και κρατάει μια αναφορά στο αντικείμενο κλήσης (το αντικείμενο που καλεί την μέθοδο).
- Την χρησιμοποιήσαμε για να διαφοροποιήσουμε τα πεδία του αντικειμένου από παραμέτρους με το ίδιο όνομα

```
class Person
{
        private String name;
        private int age;

        public Person(String name, int age){
                this.name = name;
                this.age = age;
        }
}
```

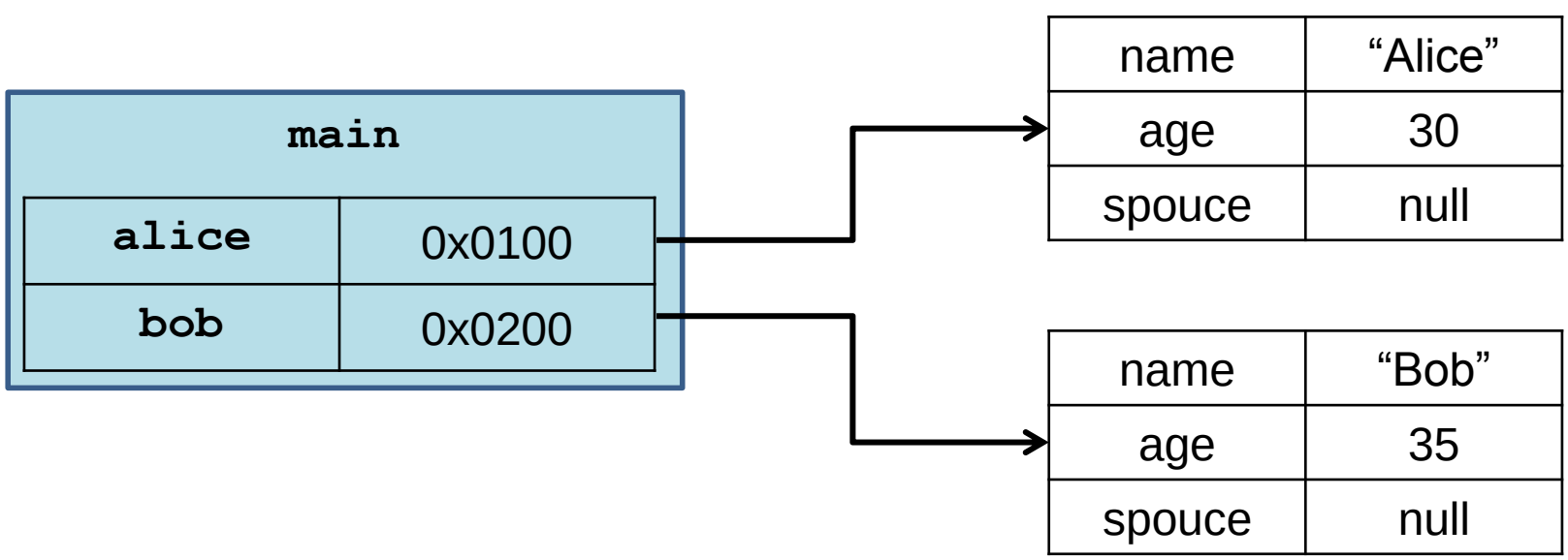
Η μεταβλητή this

- Εκτός από αυτή την χρήση, μπορούμε να χρησιμοποιήσουμε την μεταβλητή this σαν οποιαδήποτε άλλη μεταβλητή
  - Μπορούμε να την περάσουμε σαν παράμετρο σε κάποια μέθοδο
  - Μπορούμε να την αναθέσουμε σε κάποια μεταβλητή
  - Μπορούμε να την επιστρέψουμε σε κάποια μέθοδο.

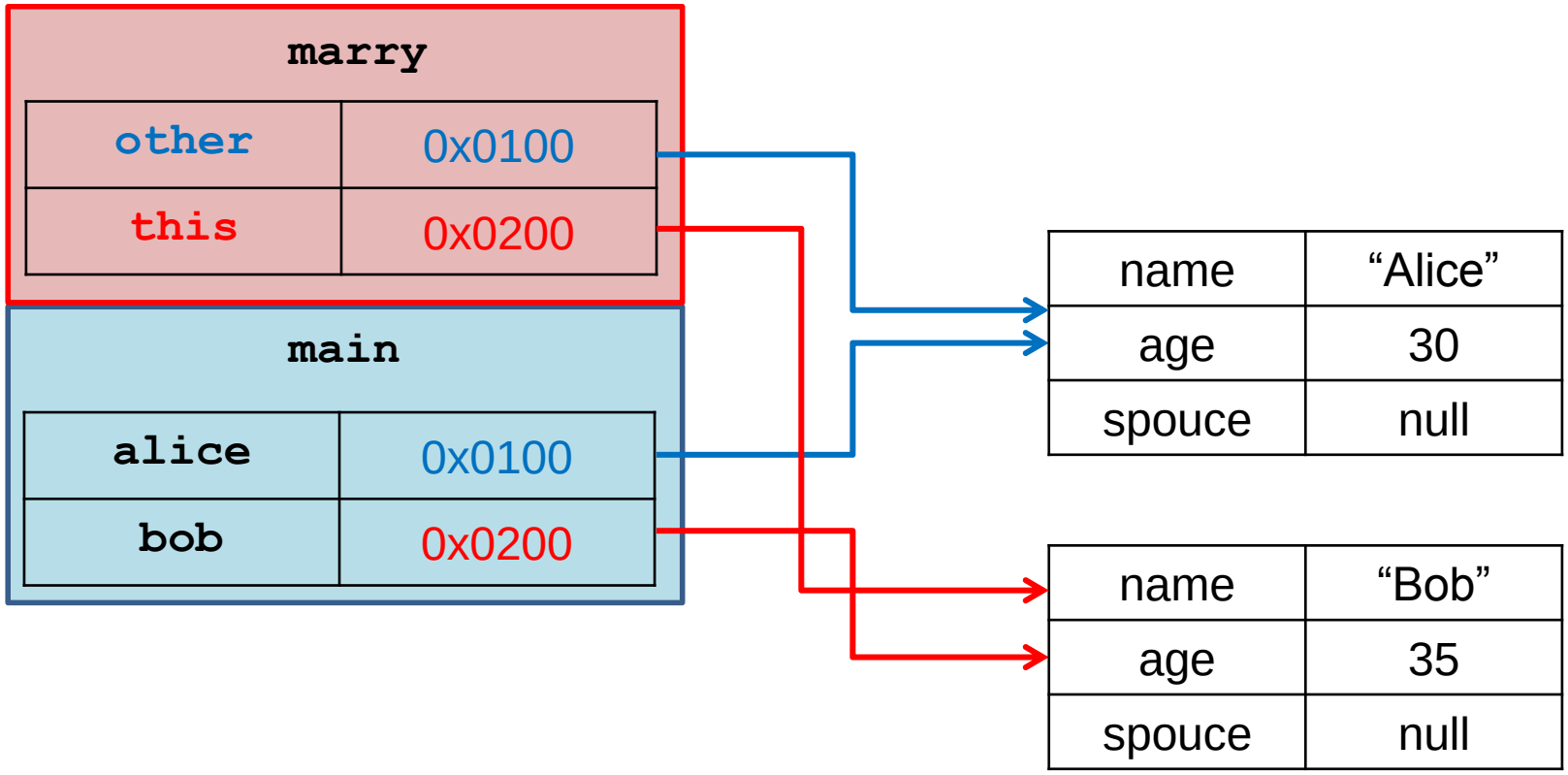
- Αυτό είναι χρήσιμο όταν χρειαζόμαστε μια αναφορά στο αντικείμενο κλήσης.

```
class Person
{
        private String name;
        private int age;
        private Person spouce;

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

        public Person getOlderPerson(Person other){
                if (this.age > other.age){
                        return this;
                }
                return other;
        }

        public void marry(Person other){
                this.spouce = other;
                other.spouce = this;
        }
}
```

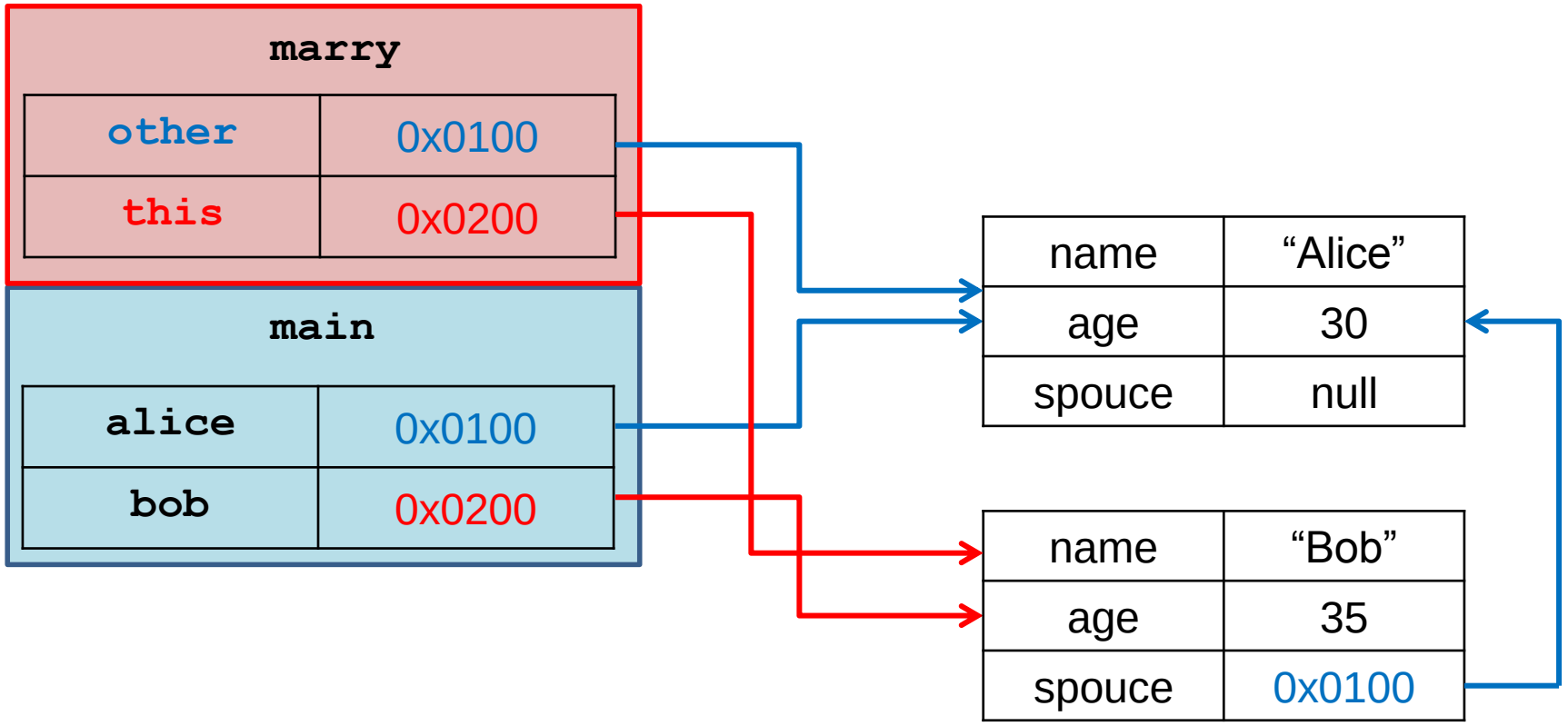
```
class Test
{
        public static void main(String[] args){
                Person alice = new Person("Alice", 30);
                Person bob = new Person("Bob", 35);
                Person older = bob.getOlderPerson(alice);
                bob.marry(alice);
}
```


main
alice
0x0100
bob
0x0200
name
“Alice”
age
30
spouce
null
name
“Bob”
age
35
spouce
null

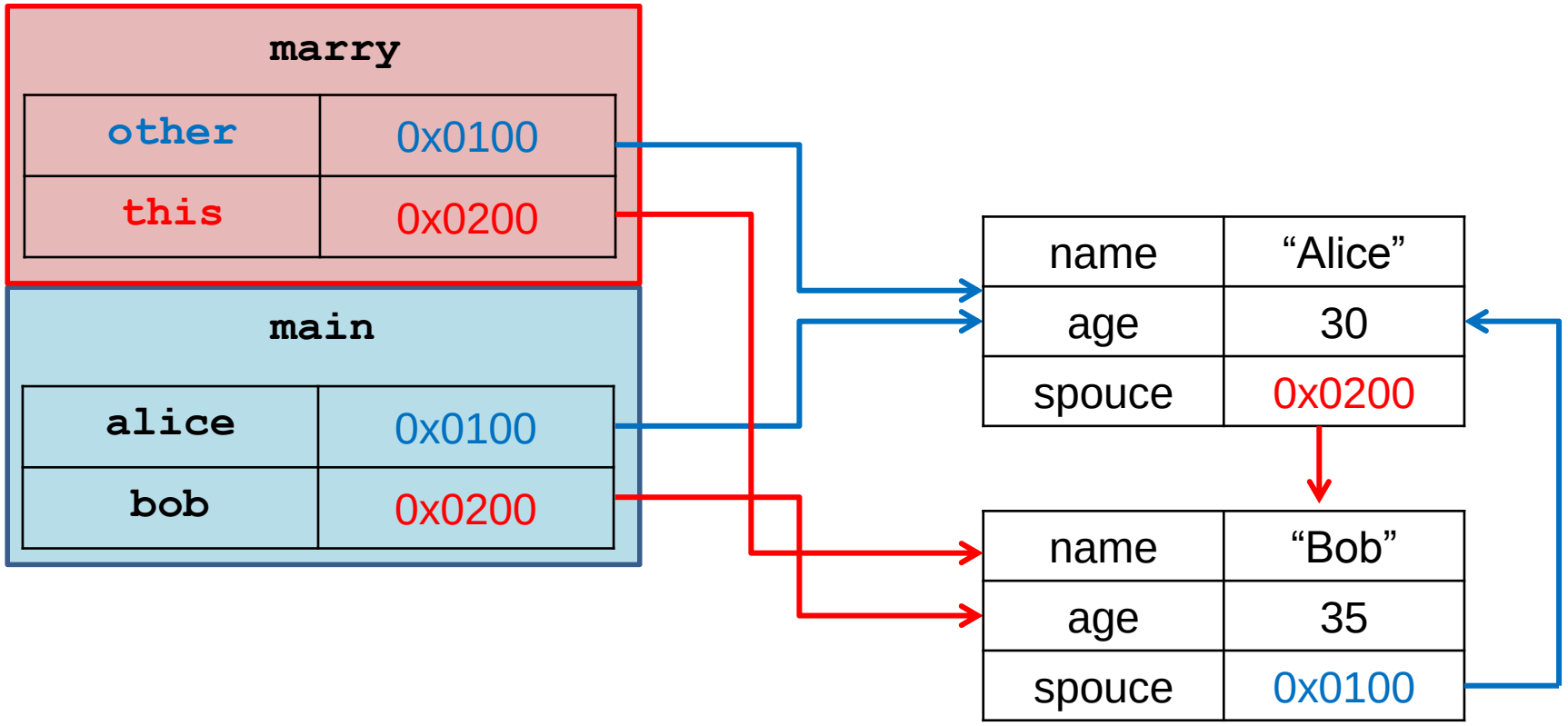
```
bob.marry(alice);
```



```
public void marry(Person other){
        this.spouce = other;
        other.spouce = this;
}
```



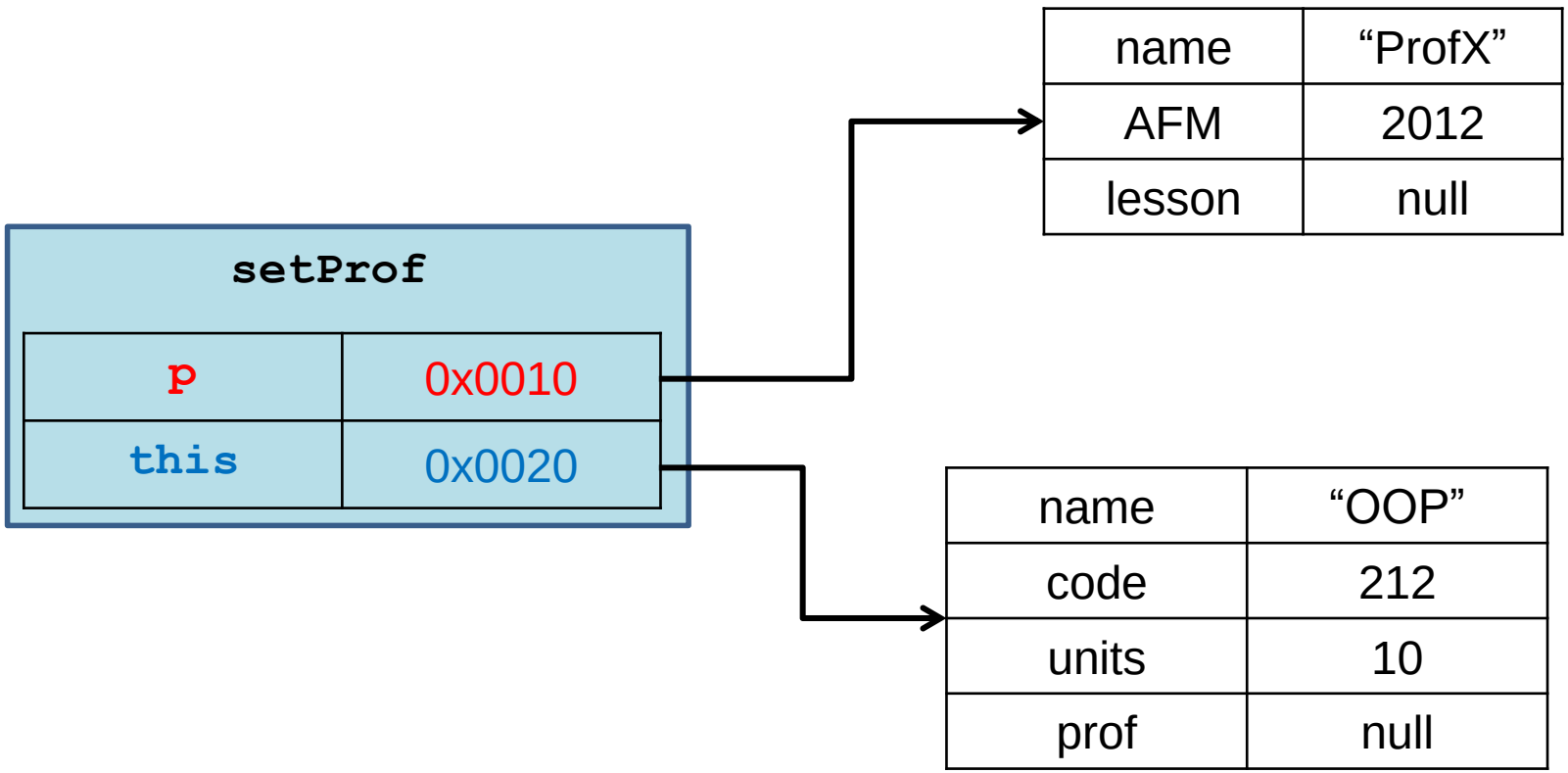
```
public void marry(Person other){
        this.spouce = other;
        other.spouce = this;
}
```



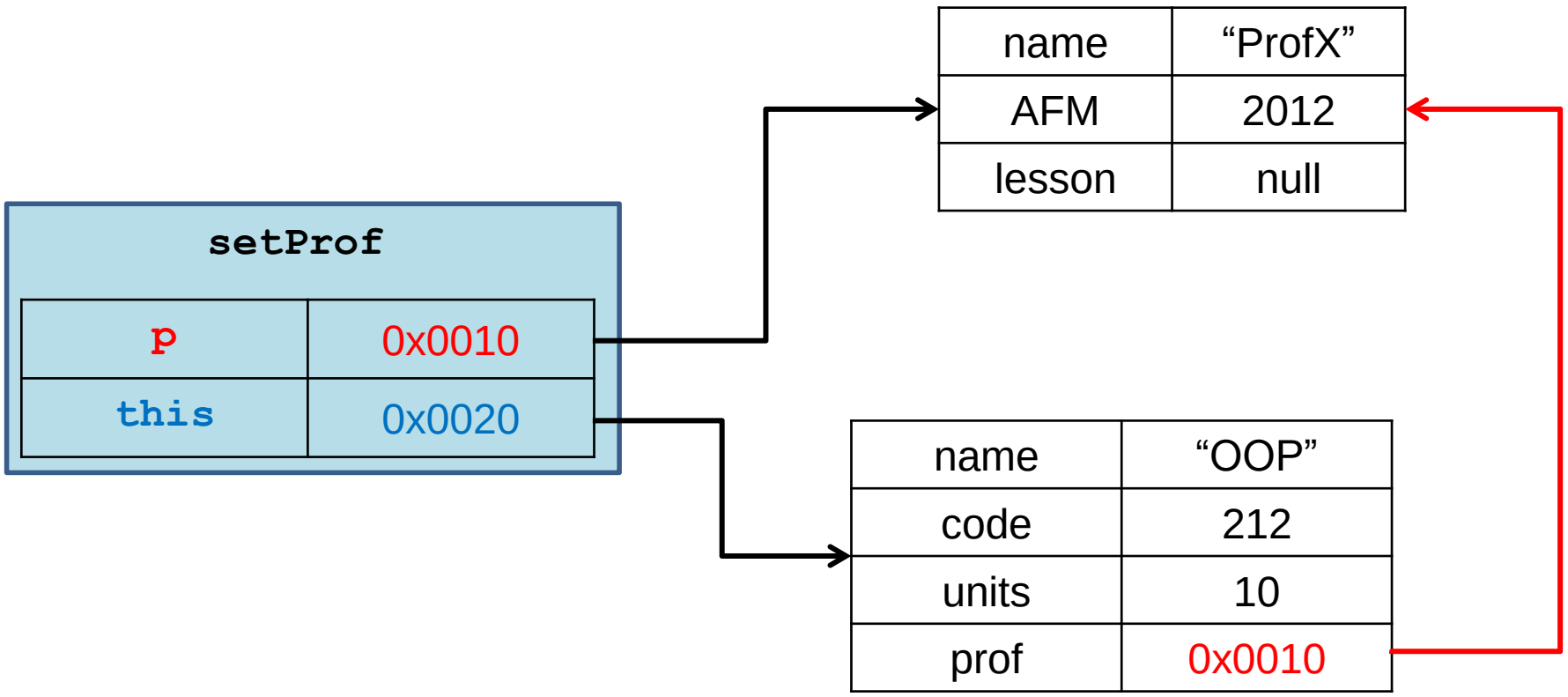
```
public class Professor
{
	private String name;
	private int AFM;
	private Course lesson;

	public Professor(String name, int afm){
		this.name = name;
		this.AFM = afm;
	}

	public void setLesson(Course c){
		lesson = c;
	}

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

```
import java.util.ArrayList;
import java.util.Scanner;

public class Course
{
        private String name;
        private int code;
        private int units;
        private Professor prof;
        private ArrayList<StudentRecord> studentList
                                = new ArrayList<StudentRecord>();

        public Course(String name, int code, int units){
                this.name = name;
                this.code = code;
                this.units = units;
        }

        public void setProf(Professor p){
                prof = p;
                p.setLesson(this);
        }

}
```

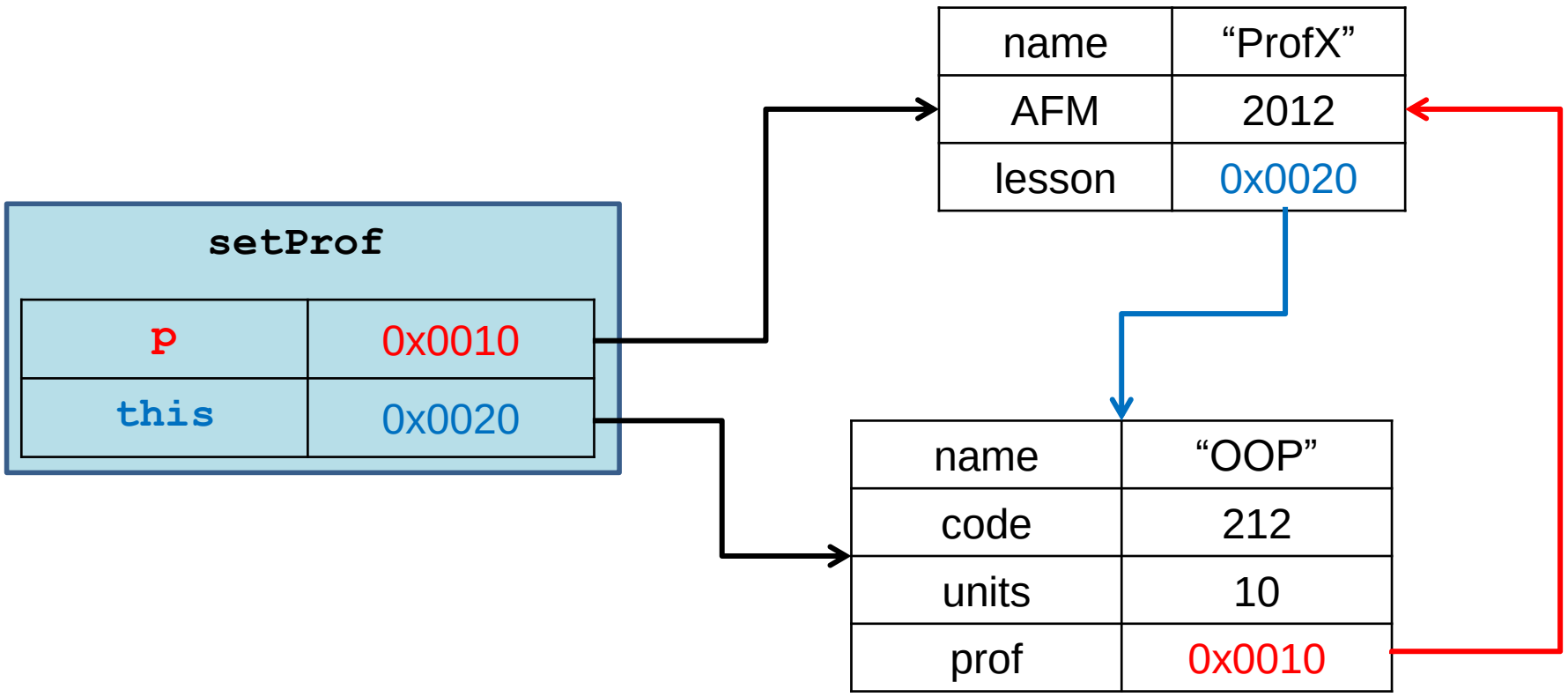
```
public void setProf(Professor p){
        prof = p;
        p.setLesson(this);
}
```



```
public void setProf(Professor p){
        prof = p;
        p.setLesson(this);
}
```



```
public void setProf(Professor p){
        prof = p;
        p.setLesson(this);
}
```



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.

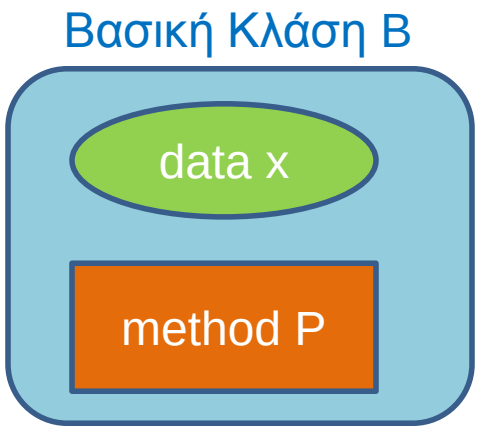
- Αυτό το καταφέρνουμε με την κληρονομικότητα!

Κληρονομικότητα

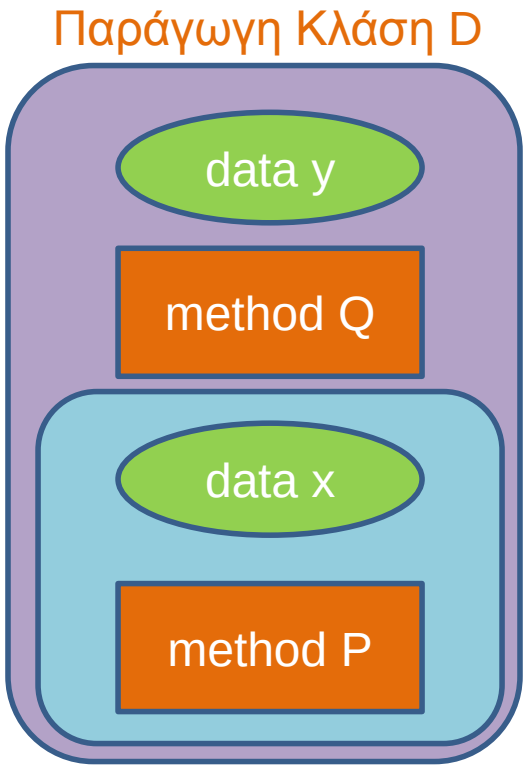
- Η κληρονομικότητα είναι κεντρική έννοια στον αντικειμενοστραφή προγραμματισμό.
- Η ιδέα είναι να ορίσουμε μια γενική κλάση που έχει κάποια χαρακτηριστικά (πεδία και μεθόδους) που θέλουμε και μετά να ορίσουμε εξειδικευμένες παραλλαγές της κλάσης αυτής στις οποίες προσθέτουμε ειδικότερα χαρακτηριστικά.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

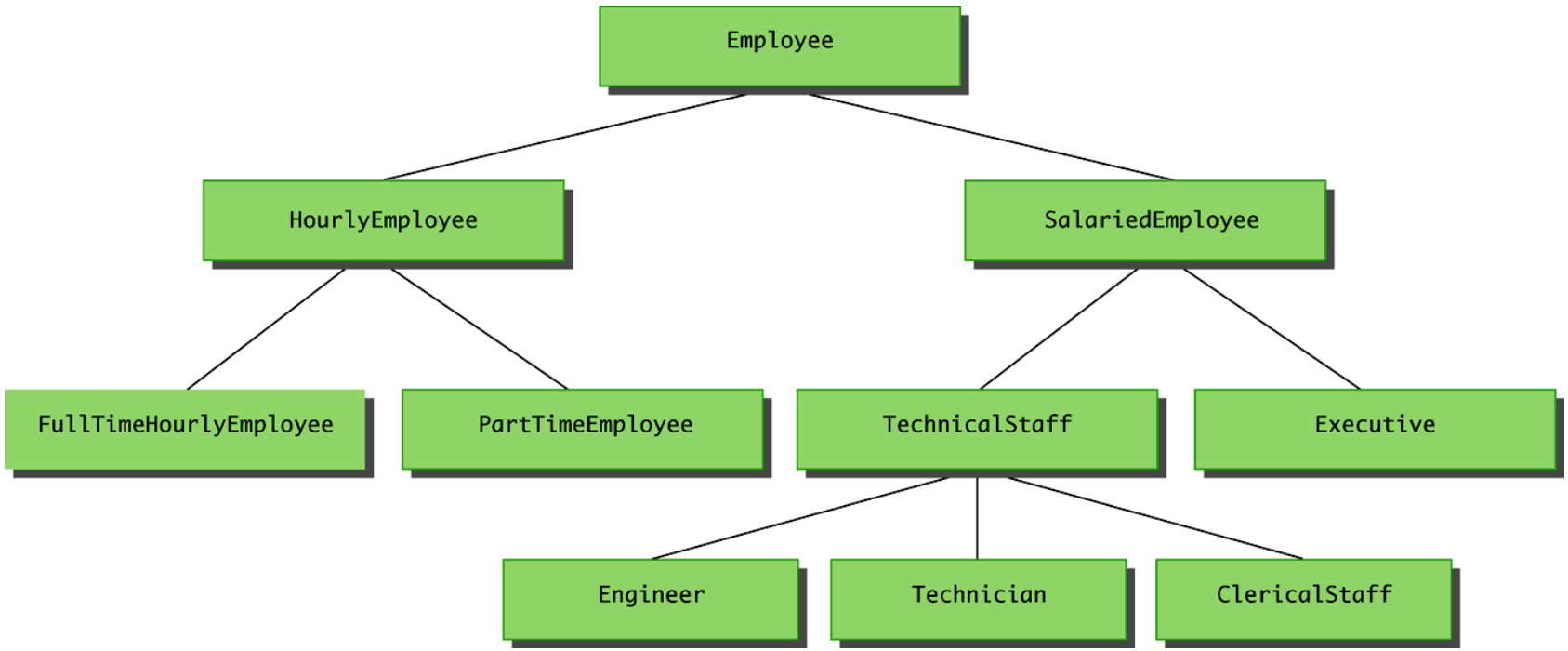
- Η κληρονομικότητα είναι χρήσιμη όταν
  - Θέλουμε να έχουμε αντικείμενα και της κλάσης B και της κλάσης D.
  - Θέλουμε να ορίσουμε πολλαπλές παράγωγες κλάσεις D1, D2, … που η κάθε μία επεκτείνει την B με διαφορετικό τρόπο.
- Μπορούμε να ορίσουμε παράγωγες κλάσεις των παράγωγων κλάσεων.
  - Με αυτό τον τρόπο ορίζεται μια ιεραρχία κλάσεων.

Ιεραρχία κλάσεων (Class Hierarchy)

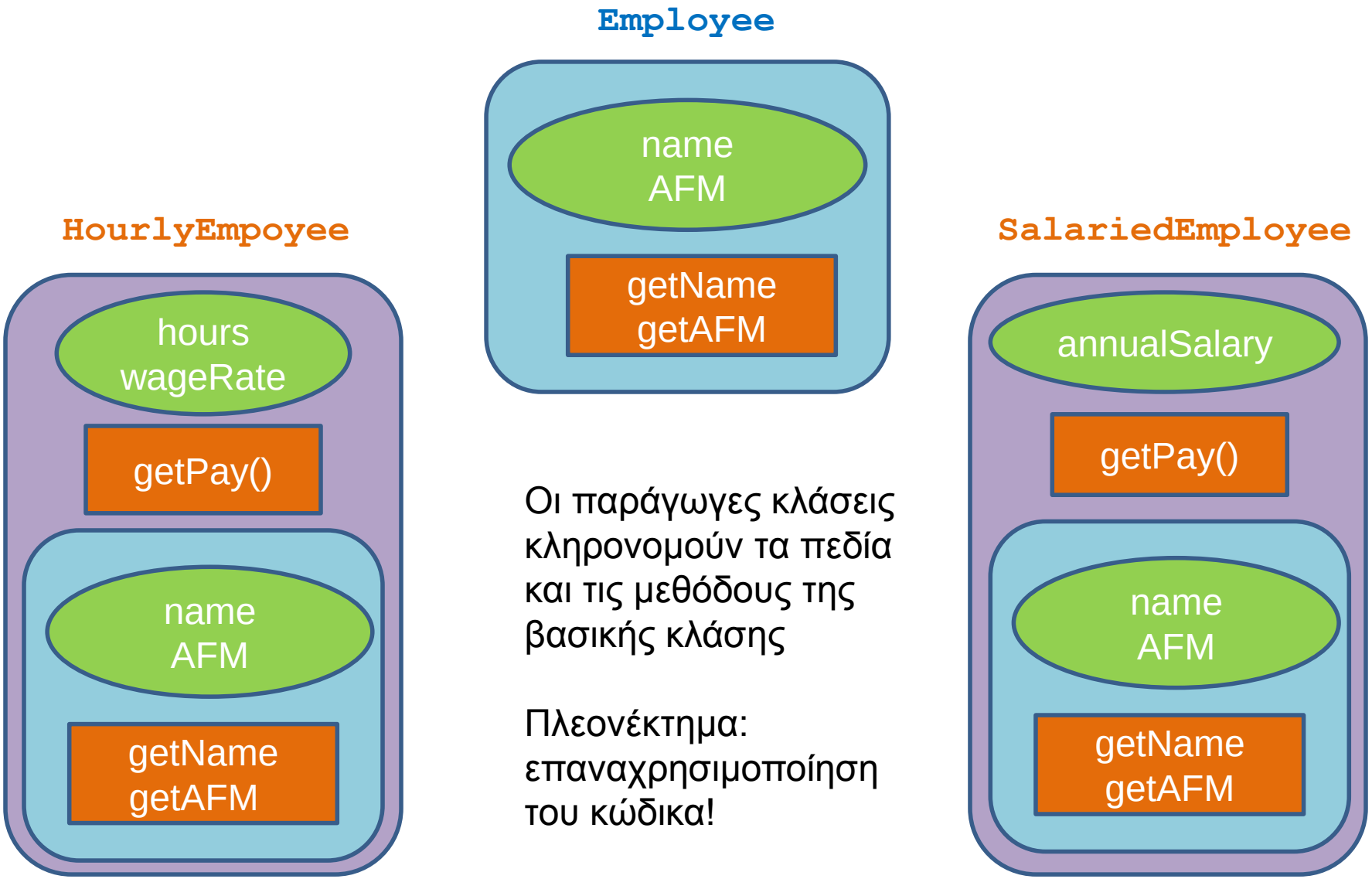
- Παράδειγμα: Έχουμε ένα πρόγραμμα που διαχειρίζεται τους Εργαζόμενους μιας εταιρίας.
  - Όλοι οι εργαζόμενοι έχουν κοινά χαρακτηριστικά το όνομα τους και το ΑΦΜ τους.
- Οι εργαζόμενοι χωρίζονται σε Ωρομίσθιους και Έμμισθους
  - Διαφορετικά χαρακτηριστικά θα κρατάμε όσον αφορά το μισθό για τον καθένα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



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

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

```
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 καλείτε 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!

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

    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 χρήσης

```
public class InheritanceDemo
{
    public static void main(String[] args)
    {
        HourlyEmployee joe = new HourlyEmployee("Joe Worker",
                                                100, 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 την μέθοδο toStrong 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",
                                                    100, 100000);
        HourlyEmployee sam = new HourlyEmployee("Sam",
                                                  200, 50.50, 40);

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

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

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

    }

    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

```
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.

```
public class IsADemo
{
    public static void main(String[] args)
    {
        SalariedEmployee joe = new SalariedEmployee("Josephine",
                                                    100, 100000);
        HourlyEmployee sam = new HourlyEmployee("Sam",
                                                200, 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 και όχι την toString της Employee.