

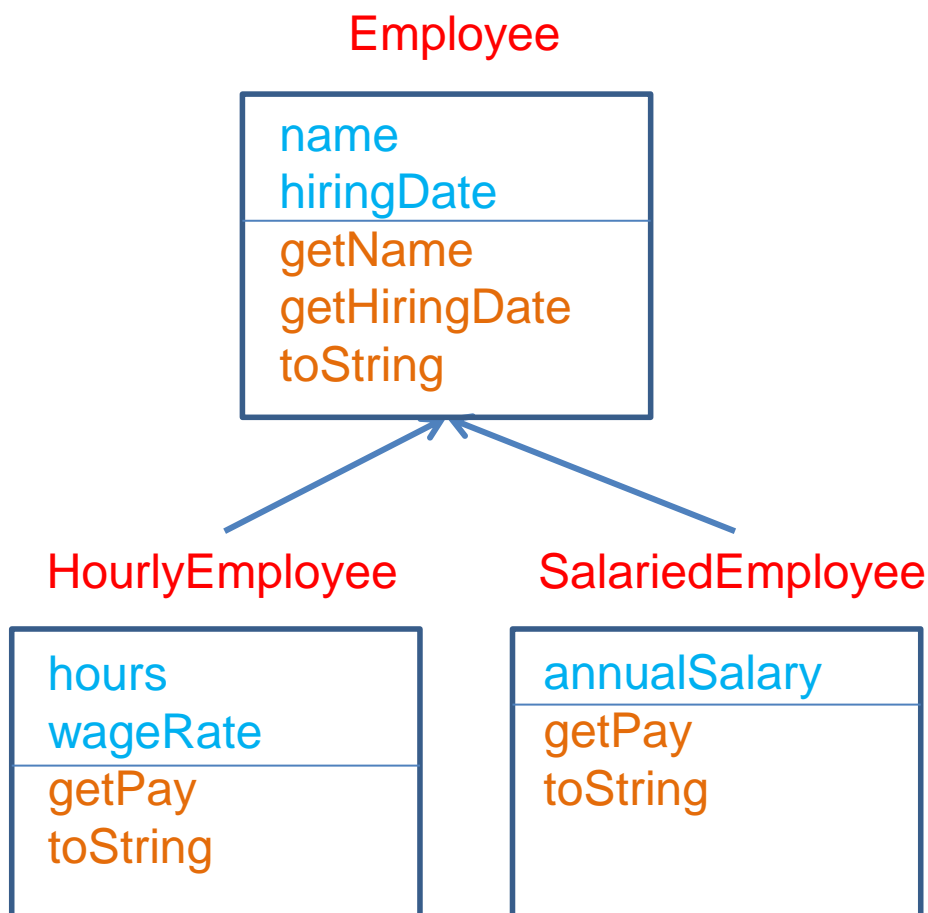
ΤΕΧΝΙΚΕΣ ΑΝΤΙΚΕΙΜΕΝΟΣΤΡΑΦΟΥΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ

Πολυμορφισμός – Late Binding

Αφηρημένες κλάσεις

Interfaces – διεπαφές

Κληρονομικότητα



Οι παράγωγες κλάσεις κληρονομούν τα πεδία και τις μεθόδους της βασικής κλάσης και έχουν και δικά τους πεδία και μεθόδους.

Επίσης μπορούμε να υπερβαίνουμε (override) κάποιες μεθόδους (`toString`)

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

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

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

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

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

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

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

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

        System.out.println("showEmployee(han) invoked:");
        showEmployee(han);
    }

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

Τι θα τυπώσει η `showEmployee` όταν την καλέσουμε με ορίσματα το `sam` και το `han`?
Ποια μέθοδος `toString` θα κληθεί?

```

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

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

        System.out.println("showEmployee(han) invoked:");
        showEmployee(han);
    }

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

```

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**)

Late Binding (καθυστερημένη δέσμευση)

- Η **δέσμευση (binding)** αναφέρεται στον συσχετισμό μεταξύ της **κλήσης μιας μεθόδου** και του **ορισμού (κώδικα) της μεθόδου**.
- **Early binding:** Η δέσμευση γίνεται **κατά τη μεταγλώττιση** του προγράμματος
 - Στην περίπτωση αυτή η μέθοδος **toString()** που θα κληθεί θα είναι η μέθοδος της κλάσης **Employee** μιας και όταν γίνεται η μεταγλώττιση ο compiler βλέπει το όρισμα ως αντικείμενο της κλάσης **Employee**.
- **Late binding:** Η δέσμευση γίνεται **κατά τη εκτέλεση** του προγράμματος
 - Το κάθε αντικείμενο έχει **πληροφορία** για την κλάση του και τον ορισμό (κώδικα) των μεθόδων του.
 - 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).

Παράδειγμα

```
public class Example3
{
    public static void main(String[] args)
    {
        Employee employeeArray[] = new Employee[3];

        employeeArray[0] = new Employee("alice",
                                         new Date(1,1,2010));

        employeeArray[1] = new HourlyEmployee("bob",
                                               new Date(1,1,2011), 20, 160);

        employeeArray[2] = new SalariedEmployee("charlie",
                                                new Date(1,1,2012), 24000);

        for (int i = 0; i < 3; i++){
            System.out.println(employeeArray[i]);
        }
    }
}
```

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

```
public class Sale
{
    protected String name;
    protected double price;

    public Sale(String theName, double thePrice){
        name = theName;
        price = thePrice;
    }

    public String toString( ){
        return (name + " Price and total cost = $" + price);
    }

    public double bill( ){
        return price;
    }

    public boolean equalDeals(Sale otherSale){
        return (name.equals(otherSale.name)
            && this.bill( ) == otherSale.bill( ));
    }

    public boolean lessThan (Sale otherSale){
        return (this.bill( ) < otherSale.bill( ));
    }
}
```

Σύμφωνα με το βιβλίο δεν συνιστάται η χρήση της `protected` αλλά την χρησιμοποιούμε για απλότητα στο παράδειγμα


```
public class DiscountSale extends Sale
{
    private double discount;

    public DiscountSale(String theName,
                        double thePrice, double theDiscount)
    {
        super(theName, thePrice);
        discount = theDiscount;
    }
}
```

```
public double bill( )
{
    double fraction = discount/100;
    return (1 - fraction)*price;
}
```

Υπέρβαση της μεθόδου `bill()`

```
public String toString( )
{
    return (name + " Price = $" + price
           + " Discount = " + discount + "%\n"
           + " Total cost = $" + bill( ));
}
```

Δεν έχουμε υπέρβαση των μεθόδων `equalDeals` και `lessThan`

```

public class LateBindingDemo
{
    public static void main(String[] args)
    {
        Sale simple = new Sale("floor mat", 10.00); //One item at $10.00.
        DiscountSale discount = new DiscountSale("floor mat", 11.00, 10);
            //One item at $11.00 with a 10% discount.

        System.out.println(simple);
        System.out.println(discount);

        if (discount.lessThan(simple))
            System.out.println("Discounted item is cheaper.");
        else
            System.out.println("Discounted item is not cheaper.");

        Sale regularPrice = new Sale("cup holder", 9.90); //One item at $9.90.
        DiscountSale specialPrice = new DiscountSale("cup holder", 11.00, 10);
            //One item at $11.00 with a 10% discount.

        System.out.println(regularPrice);
        System.out.println(specialPrice);

        if (specialPrice.equalDeals(regularPrice))
            System.out.println("Deals are equal.");
        else
            System.out.println("Deals are not equal.");
    }
}

```

Οι **lessThan** και **equalDeals** κληρονομούνται από την **Sale**

Με το μηχανισμό του **late binding** στην κλήση τους ξέρουμε ότι το αντικείμενο που τις καλεί είναι τύπου **DiscountSale**

Ξέρουμε λοιπόν ότι όταν εκτελούμε τον κώδικα της **lessThan** και **equalDeals** η μέθοδος **bill()** που θα πρέπει να καλέσουμε είναι αυτή της **DiscountSale** ενώ για το **otherSale.bill()** είναι αυτή της **Sale**

Ένα διαφορετικό πρόβλημα

- Ας υποθέσουμε ότι στην **Employee** θέλουμε να προσθέσουμε μια μέθοδο που ελέγχει αν δύο υπάλληλοι έχουν τον ίδιο μισθό (ανεξάρτητα αν είναι ωρομίσθιοι, ή πλήρους απασχόλησης)
- Η συνάρτηση είναι απλή:

```
public boolean sameSalary(Employee other)
{
    if (this.getPay() == other.getPay()) {
        return true;
    }
    return false
}
```

- Το **πρόβλημα**: Που θα την ορίσουμε?
 - Ιδανικά στην **Employee**, αλλά η **Employee** δεν έχει συνάρτηση **getPay()**
 - Αν την ορίσουμε στην **HourlyEmployee**, ή στην **SalariedEmployee**, δεν μπορούμε να περάσουμε όρισμα **Employee** εφόσον δεν έχει μέθοδο **getPay()**

Αφηρημένες μέθοδοι

- Η λύση είναι να ορίσουμε την `getPay()` ως αφηρημένη μέθοδο (`abstract method`) της `Employee`.
 - `public abstract double getPay();`
 - Μια αφηρημένη μέθοδος **δηλώνεται** σε μία κλάση αλλά **ορίζεται** στις παράγωγες κλάσεις.
 - Χρησιμοποιούμε τη δεσμευμένη λέξη **abstract** για να δηλώσουμε ότι μια μέθοδος είναι αφηρημένη.
 - Η δήλωση μιας αφηρημένης μεθόδου δεν έχει κώδικα οπότε η εντολή τερματίζει με το **;**
 - Οι αφηρημένες μέθοδοι πρέπει να είναι **public** (ή `protected`), όχι `private`.

Αφηρημένες κλάσεις

- Οι κλάσεις που περιέχουν μια αφηρημένη μέθοδο ορίζονται **υποχρεωτικά** ως **αφηρημένες κλάσεις** (**abstract classes**)
 - `public abstract class Employee { ... }`
- **Δεν μπορούμε** να δημιουργήσουμε **αντικείμενα** μιας **αφηρημένης κλάσης**
 - Μια αφηρημένη κλάση χρησιμοποιείται μόνο για να δημιουργούμε **παράγωγες κλάσεις**.
 - Στην περίπτωση μας δεν χρειαζόμαστε αντικείμενα τύπου Employee. Ένας υπάλληλος θα είναι είτε ωρομίσθιος, είτε μόνιμος.
- Οι **παράγωγες** κλάσεις μιας αφηρημένης κλάσης θα **πρέπει πάντα** να **ορίζουν** τις **αφηρημένες μεθόδους**
 - **Εκτός** αν είναι και αυτές **αφηρημένες**.
- Μια κλάση (ή μέθοδος) που δεν είναι αφηρημένη λέγεται **ενυπόστατη** (**concrete**)

```
public abstract class Employee
```

Ορισμός της αφηρημένης κλάσης

```
{
```

```
    private String name;  
    private Date hireDate;
```

Ορισμός της αφηρημένης μεθόδου

```
    public abstract double getPay();
```

```
    public boolean samePay(Employee other) {  
        return (this.getPay() == other.getPay());  
    }
```

Χρήση της αφηρημένης μεθόδου
και της αφηρημένης κλάσης

```
    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()
```

Όταν καλέσουμε την **samePay** θα την καλέσουμε με ένα αντικείμενο μιας από τις παράγωγες κλάσεις.

```
}
```

```
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( ){ ... }
}
```

Εφόσον η κλάση HourlyEmployee παράγεται από αφηρημένη κλάση και η ίδια δεν είναι αφηρημένη, πρέπει **υποχρεωτικά** να ορίσει την αφηρημένη μέθοδο getPay

```
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( ) { ... }
}
```

Εφόσον η κλάση SalariedEmployee παράγεται από αφηρημένη κλάση και η ίδια δεν είναι αφηρημένη, πρέπει **υποχρεωτικά** να ορίσει την αφηρημένη μέθοδο getPay


```
public class Example
{
    public static void main(String args[]) {
        HourlyEmployee A = new HourlyEmployee("Alice",
            new Date(4,18,2013), 10, 100);
        SalariedEmployee B = new SalariedEmployee("Bob",
            new Date(4,17,2013), 12000);
        if (A.samePay(B)) {
            System.out.println("The two employees
                earn the same amount per month");
        }
        else{
            System.out.println("The two employees do NOT
                earn the same amount per month");
        }
    }
}
```

```

public class Example
{
    public static void main(String args[]) {
        Employee A = new HourlyEmployee("Alice",
            new Date(4,18,2013), 10, 100);
        Employee B = new SalariedEmployee("Bob",
            new Date(4,17,2013), 12000);
        if (A.samePay(B)) {
            System.out.println("The two employees
                earn the same amount per month");
        }
        else{
            System.out.println("The two employees do NOT
                earn the same amount per month");
        }
    }
}

```

Μπορούμε να ορίσουμε **μεταβλητές αφηρημένης κλάσης**. Θα πρέπει να όμως να τους αναθέσουμε **αντικείμενα** μιας από τις **παράγωγες ενυπόστατες κλάσεις**. Δεν μπορούμε να ορίσουμε ένα αντικείμενο της αφηρημένης κλάσης.

```
public class Example
{
    public static void main(String args[]){
        HourlyEmployee A = new HourlyEmployee("Alice",
            new Date(4,18,2013), 10, 100);
        SalariedEmployee B = new SalariedEmployee("Bob",
            new Date(4,17,2013), 12000);
        compareAndPrint(A,B)
    }

    private static void compareAndPrint(Employee A, Employee B){
        if (A.samePay(B)){
            System.out.println("The two employees
                earn the same amount per month");
        }
        else{
            System.out.println("The two employees do NOT
                earn the same amount per month");
        }
    }
}
```

Αφηρημένες κλάσεις

- **Αφηρημένες κλάσεις** είναι οι κλάσεις που περιέχουν **αφηρημένες μεθόδους**
 - Η **υλοποίηση** των αφηρημένων μεθόδων μετατίθεται στις μη αφηρημένες (**ενυπόστατες – concrete**) κλάσεις που είναι **απόγονοι** μιας **αφηρημένης κλάσης**.
 - Η υλοποίηση είναι **υποχρεωτική**. Άρα έτσι εξασφαλίζουμε ότι μια concrete κλάση θα έχει την μέθοδο που θέλουμε.
- Οι αφηρημένες κλάσεις εκτός από αφηρημένες μεθόδους έχουν και **πεδία** και **ενυπόστατες μεθόδους**.
 - Κληρονομούν επιπλέον **χαρακτηριστικά** στους απογόνους τους, όχι μόνο τις αφηρημένες μεθόδους.

Interfaces

- Ένα **interface** είναι μια ακραία μορφή αφηρημένης κλάσης
 - Ένα interface έχει **μόνο δηλώσεις** μεθόδων.
 - Το interface ορίζει μια **απαραίτητη λειτουργικότητα** που θέλουμε.

Παραδείγματα

```
public interface MovingObject
{
    public void move();
}
```

```
public interface ElectricObject
{
    public boolean powerOn();

    public boolean powerOff();
}
```

Interfaces

- Μία κλάση υλοποιεί το interface.
 - Η κλάση μπορεί να είναι και αφηρημένη κλάση
- Μια κλάση μπορεί να υλοποιεί πολλαπλά interfaces
 - Αλλά δεν μπορεί να κληρονομεί από πολλαπλές κλάσεις

Παραδείγματα

```
public class Car implements MovingObject
```

```
{
```

```
    ...
```

```
}
```

```
public class ElectricCar
```

```
    implements MovingObject, ElectricObject
```

```
{    ...    }
```

```
public abstract class Vehicle implements MovingObject
```

```
{
```

```
    public abstract void move();
```

```
}
```

```
public class ElectricCar
```

```
    extends Vehicle, implements ElectricObject
```

```
{    ...    }
```


Interfaces

- Ένα Interface μπορεί να κληρονομεί από ένα άλλο interface

```
public interface ElectricMovingObject
    extends MovingObject
{
    public boolean powerOn();

    public boolean powerOff();
}
```

Interfaces vs αφηρημένες κλάσεις

- Τα **interfaces** είναι χρήσιμα όταν θέλουμε να ορίσουμε αντικείμενα που ορίζονται μόνο από κάποια **υψηλού επιπέδου λειτουργικότητα** ενώ κατά τα άλλα μπορεί να είναι πολύ διαφορετικά μεταξύ τους
 - Έχουν το ίδιο interface – ένα κινούμενο αντικείμενο μπορεί να κινείται
 - Δεν ξέρουμε πως, σε πόσες διαστάσεις, με τι ταχύτητα κλπ.
- Μια **αφηρημένη κλάση** υποθέτει ότι τα αντικείμενα που θα ορίσουμε έχουν πολλά περισσότερα **κοινά χαρακτηριστικά**
 - Κοινά πεδία πάνω στα οποία μπορούμε να υλοποιήσουμε και κοινές μεθόδους.

Αφηρημένοι Τύποι Δεδομένων

- Τα interfaces μπορούμε να τα δούμε και σαν **Αφηρημένους Τύπους Δεδομένων**
- Π.χ., μία **στοίβα** απαιτεί συγκεκριμένες λειτουργίες από τις κλάσεις που την υλοποιούν
 - Push
 - Pop
 - IsEmpty
 - Top
- Ανάλογα με τον τύπο των δεδομένων που θα κρατάει η στοίβα μπορούμε να ορίσουμε διαφορετικές **υλοποιήσεις**
 - Υπάρχει και άλλος τρόπος να το κάνουμε αυτό όμως όπως θα δούμε παρακάτω

Παράδειγμα: Το interface myComparable

- Το interface **myComparable** ορίζει interface για αντικείμενα τα οποία μπορούν να **συγκριθούν** μεταξύ τους
 - Υπάρχει στην Java το interface Comparable αλλά είναι λίγο διαφορετικό
- Ορίζει την μέθοδο
 - `public int compareTo(Object other);`
- Σημασιολογία:
 - Αν η μέθοδος επιστρέψει **αρνητικό αριθμό** τότε το αντικείμενο **this** είναι **μικρότερο** από το αντικείμενο **other**
 - Αν η μέθοδος επιστρέψει **μηδέν** τότε το αντικείμενο **this** είναι **ίσο** με το αντικείμενο **other**
 - Αν η μέθοδος επιστρέψει **θετικό αριθμό** τότε το αντικείμενο **this** είναι **μεγαλύτερο** από το αντικείμενο **other**

Interface myComparable

```
public interface myComparable
{
    public int compareTo(myComparable other);
}
```

Εφαρμογή

- Μπορούμε να ορίσουμε μια μέθοδο `sort` η οποία να μπορεί να εφαρμοστεί σε πίνακες με οποιαδήποτε μορφής αντικείμενα

```
public static void sort(myComparable[] array) {  
    for (int i = 0; i < array.length; i ++){  
        myComparable minElement = array[i];  
        for (int j = i+1; j < array.length; j ++){  
            if (minElement.compareTo(array[j]) > 0){  
                minElement = array[j];  
                array[j] = array[i];  
                array[i] = minElement;  
            }  
        }  
    }  
}
```

Μπορεί να εφαρμοστεί σε **οποιαδήποτε** αντικείμενα που υλοποιούν το interface `myComparable`

```
import java.util.Scanner;

class Person implements myComparable
{
    private String name;
    private int number;

    public Person() {
        System.out.println("enter name and number:");
        Scanner input = new Scanner(System.in);
        name = input.next(); number = input.nextInt();
    }

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

    public int compareTo(myComparable other) {
        Person otherPerson = (Person) other;
        if (number < otherPerson.number) {
            return -1;
        } else if (number == otherPerson.number) {
            return 0;
        } else { return 1;}
    }
}
```

Χρήση του DownCasting

```
public class ComparableExample
{
    public static void main(String[] args){
        Person[] array = new Person[5];
        for (int i = 0; i < array.length; i ++){
            array[i] = new Person();
        }
        sort(array);
        System.out.println();
        for (int i = 0; i < array.length; i ++){
            System.out.println(array[i]);
        }
    }

    public static void sort(myComparable[] array){
        for (int i = 0; i < array.length; i ++){
            myComparable minElement = array[i];
            for (int j = i+1; j < array.length; j ++){
                if (minElement.compareTo(array[j]) > 0){
                    minElement = array[j];
                    array[j] = array[i];
                    array[i] = minElement;
                }
            }
        }
    }
}
```


Επέκταση

- Τι γίνεται αν αντί για Persons θέλουμε να συγκρίνουμε σπίτια?
 - Ένα σπίτι (House) έχει διεύθυνση και μέγεθος
 - Θέλουμε να ταξινομήσουμε με βάση το μέγεθος