ΤΕΧΝΙΚΕΣ ΑΝΤΙΚΕΙΜΕΝΟΣΤΡΑΦΟΥΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ

Κληρονομικότητα

Downcasting

Πολυμορφισμός – Late Binding

toString και equals

- Είπαμε ότι 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 το κλάσεων και κάθε νέα κλάση μπορεί να τις υπερβεί (override).
  - Είδαμε παραδείγματα πως υπερβήκαμε την μέθοδο toString.

equals

- Η equals στην κλάση Object ορίζεται ως:
  - `public boolean equals(Object other)`
- Για την κλάση Employee θα την ορίσουμε ως:
  - `public boolean equals(Employee other)`
- Αλλάζουμε την υπογραφή της κλάσης, άρα δεν κάνουμε υπέρβαση, αλλά υπερφόρτωση της equals
  - Πως θα την ορίσουμε ώστε να κάνουμε υπέρβαση?

Overriding equals

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

    public boolean equals(Object otherObject)
    {
        if (otherObject == null)
            return false;
        else if (getClass( ) != otherObject.getClass( ))
            return false;
        else
        {
            Employee otherEmployee = (Employee)otherObject;
            return (name.equals(otherEmployee.name)
               && hireDate.equals(otherEmployee.hireDate));
        }
    }
}
```

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 δεν γίνει σωστά μπορεί να προκαλέσει λάθη κατά την εκτέλεση του προγράμματος

Downcasting

```
public class DowncastingExample
{
    public static void main(String[] args)
    {
                SalariedEmployee sam = new SalariedEmployee("Sam",
                                new Date(1, 1, 2010), 100000);
                Employee eve = new Employee("Eve", new Date(1,1,2012));

                SalariedEmployee eve2 = eve;
                if (sam.getHireDate().equals(eve2.getHireDate())){
                        System.out.println("Same hire date");
                }else{
                        System.out.println("Different hire date");
                }
        }
}
```

Στην περίπτωση αυτή προσπαθούμε να κάνουμε το downcasting έμμεσα, αναθέτοντας μια μεταβλητή Employee σε μια μεταβλητή SalariedEmployee.
Θα μας χτυπήσει λάθος κατά την μεταγλώτιση.

Downcasting

```
public class DowncastingExample
{
    public static void main(String[] args)
    {
                SalariedEmployee sam = new SalariedEmployee("Sam",
                                new Date(1, 1, 2010), 100000);
                Employee eve = new Employee("Eve", new Date(1,1,2012));

                SalariedEmployee eve2 = (SalariedEmployee)eve;
                if (sam.getHireDate().equals(eve2.getHireDate())){
                        System.out.println("Same hire date");
                }else{
                        System.out.println("Different hire date");
                }
        }
}
```

Στην περίπτωση αυτή θα μας χτυπήσει λάθος στο τρέξιμο παρότι χρησιμοποιούμε μόνο την κοινή μέθοδο getHireDate(). Το πρόγραμμα προβλέπει ότι μπορεί να υπάρχει πρόβλημα. Δεν γίνεται να μετατρέψουμε έναν Employee σε SalariedEmployee (ο Employee δεν έχει όλα τα πεδία που χρειάζεται ένας SalariedEmployee)

Downcasting

```
public class DowncastingExample
{
    public static void main(String[] args)
    {
                SalariedEmployee sam = new SalariedEmployee("Sam",
                                    new Date(1, 1, 2010), 100000);
                Employee eve = new Employee("Eve", new Date(1,1,2012));

                method(sam,sam);

        }

        private static void method(SalariedEmployee sEmp, Employee emp){
                SalariedEmployee sEmp2 = (SalariedEmployee) emp;

                if (sEmp.getHireDate().equals(sEmp2.getSalary())){
                        System.out.println("Same Salary");
                }else{
                        System.out.println("Different salary");
                }
        }
}
```

Στην περίπτωση αυτή το downcasting δεν χτυπάει λάθος γιατί υπάρχει η δυνατότητα να καλέσουμε σωστά την μέθοδο με SalariedEmployee αντικείμενο

Downcasting

```
public class DowncastingExample
{
    public static void main(String[] args)
    {
                SalariedEmployee sam = new SalariedEmployee("Sam",
                                    new Date(1, 1, 2010), 100000);
                Employee eve = new Employee("Eve", new Date(1,1,2012));

                method(sam,eve);

        }

        private static void method(SalariedEmployee sEmp, Employee emp){
                SalariedEmployee sEmp2 = (SalariedEmployee) emp;

                if (sEmp.getHireDate().equals(sEmp2.getSalary())){
                        System.out.println("Same Salary");
                }else{
                        System.out.println("Different salary");
                }
        }
}
```

Αν όμως την καλέσουμε με αντικείμενο Employee θα πάρουμε λάθος

```
import java.util.Random;

public class DowncastingExample2
{
    public static void main(String[] args)
    {
        SalariedEmployee[] sEmployees = new SalariedEmployee[4];
        sEmployees[0] = new SalariedEmployee("employee 100",new Date(1,1,2015),1000);
        sEmployees[1] = new SalariedEmployee("employee 101",new Date(2,1,2015),2000);
        sEmployees[2] = new SalariedEmployee("employee 102",new Date(3,1,2015),3000);
        sEmployees[3] = new SalariedEmployee("employee 103",new Date(4,1,2015),4000);

        SalariedEmployee rand = (SalariedEmployee)randomSelection(sEmployees);
        System.out.println(rand);
        System.out.println("Salary per month " + rand.getPay());
    }

    private static Employee randomSelection(Employee[] employees){
        Random rndGen = new Random();
        int r = rndGen.nextInt(employees.length);
        return employees[r];
    }
}
```

Σε τι μας χρειάζεται το downcasting?

Θέλουμε να καλέσουμε την μέθοδο getPay για τυπώσουμε τον μηνιαίο μισθό. Χρειαζόμαστε downcasting

Έχουμε μια γενική μέθοδο randomSelection που επιλέγει ένα τυχαίο στοιχείο από ένα πίνακα με Employee. Θέλουμε να την χρησιμοποιήσουμε σε ένα πίνακα με SalariedEmployee

Upcasting

- Η ανάθεση στην αντίθετη κατεύθυνση (upcasting) μπορεί να γίνει χωρίς να χρειάζεται casting
  - 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)

- Για παράδειγμα, ο παρακάτω κώδικας δουλεύει χωρίς πρόβλημα:
  - `Employee anEmployee;`
  - `Hourly Employee hEmployee = new HourlyEmployee();`
  - `anEmployee = hEmployee;`

```
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("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( ));
    }
}
```

Όταν καλούμε την **showEmployee** έμμεσα κάνουμε τις αναθέσεις:

```
employeeObject = joe
employeeObject = sam
```

```
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("showEmployee(joe) invoked:");
        showEmployee(joe);

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

    }

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

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

```
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("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** 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 mySale
{
    protected String name;
    protected double price;

    public mySale(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(mySale otherSale){
        return (name.equals(otherSale.name)
                && this.bill( ) == otherSale.bill( ));
    }

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

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

```
public class myDiscountSale extends mySale
{
    private double discount;

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

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


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

Υπέρβαση της μεθόδου **bill()**

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

```
public class myLateBindingDemo
{
    public static void main(String[] args)
    {
        mySale simple = new mySale("floor mat", 10.00);//One item at $10.00.
        myDiscountSale discount = new myDiscountSale("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.");

        mySale regularPrice = new mySale("cup holder", 9.90);//One item at $9.90.
        myDiscountSale specialPrice = new myDiscountSale("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 **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 της **lessThan** και **equalDeals** η μέθοδος **bill()** που θα πρέπει να καλέσουμε είναι αυτή της **myDiscountSale** ενώ για το **otherSale.bill()** είναι αυτή της **mySale**