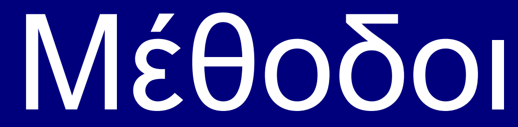
2. Μέθοδοι

Περιεχόμενα

- Μέθοδοι
  - Πέρασμα αντικειμένου σε μέθοδο
  - Υπερφόρτωση μεθόδων
  - Μέθοδοι κατασκευαστές
  - Υπερφόρτωση κατασκευαστών
- Επαναχρησιμοποίηση κώδικα
  - Σύνθεση και κληρονομικότητα
  - Πολυμορφισμός
- Αφηρημένες κλάσεις
- Διεπαφές



Μέθοδοι

Δομή μεθόδου

```
πρόσβαση τύποςΕπιστροφής όνομαΜεθόδου (τυπος1 παράμετρος1, …)
{
…
}

public double addDouble (double num1, double num2)
{
   return num1+num2;
}
```

Κλήση και αποτέλεσμα

- Όταν καλούμε μια μέθοδο αυτή
  - είτε επιστρέφει μια τιμή

  π.χ. public String getName(){…}
  - είτε δεν επιστρέφει τίποτε

  π.χ. public void setName(String n){…}
- Για να καλέσουμε μια μέθοδο, (συνήθως) χρειαζόμαστε ένα αντικείμενο της αντίστοιχης κλάσης
  - Human h1=new Human();
  - h1.setName("John");
- Όταν η μέθοδος επιστρέφει τιμή τότε πρέπει να την κρατάμε σε μια προσωρινή μεταβλητή. Διαφορετικά η επιστρεφόμενη τιμή χάνεται
  - String temp = h1.getName();

static μέθοδοι

- Καλούνται απευθείας μέσω της κλάσης

ή μέσω αντικειμένων της κλάσης

- Οι static μέθοδοι έχουν πρόσβαση στα static μέλη της τάξης

Πέρασμα αντικειμένου σε μέθοδο

- Όταν περνάμε ένα αντικείμενο σε μια μέθοδο, το πέρασμα γίνεται με αναφορά στο πραγματικό αντικείμενο.

```
public void giveName(Human h, String n){
    h.setName(n);
}
```

- Όποιες αλλαγές γίνουν στο αντικείμενο εντός της μεθόδου, περνούν μόνιμα στο αντικείμενο

```
        Human nonos=new Human();
        Human paidi=new Human();
        nonos.giveName(paidi, “Mary”);
```

Κατασκευαστές

Μέθοδος κατασκευαστής

- Βασικός ή κενός κατασκευαστής:
  - Μια public μέθοδος με όνομα ίδιο με αυτό της τάξης, χωρίς παραμέτρους και τύπο επιστροφής.
  - public Human(){ } – Δεσμεύει μνήμη για το αντικείμενο και κάνει τις αρχικοποιήσεις
  - Αν τον ορίσουμε στην τάξη, μπορούμε να ορίσουμε τις αρχικοποιήσεις που θα κάνει
  - Μπορούμε να ορίσουμε άλλους κατασκευαστές. Οπότε αναιρείται ο βασικός. Αν θέλουμε και το βασικό κατασκευαστή πρέπει υποχρεωτικά να τον ορίσουμε.

Παράδειγμα (1)

- Ορισμός ενός καλύτερου κατασκευαστή για τη Human

```
public Human (String tempName, String tempSurname, int tempAge)
{
        name=tempName;
        surname=tempSurname;
        age=tempAge;
}
```

- Αν ορίσουμε μόνο αυτόν τον κατασκευαστή τότε στη demo θα μπορούμε να χρησιμοποιούμε μόνο αυτόν.

```
        Human h1=new Human(); // βγάζει λάθος
```

Παράδειγμα (2)

- Ορισμός του βασικού κατασκευαστή στη Human

```
public Human(){
        name="";
        surname="";
        age=0;
}
```

- Διαφορετικά τα name και surname σε κάθε νέο Human είναι null.

```
        Human h1=new Human();
        System.out.println(h1.name);  //τυπώνει null
```

Η λέξη **this**

- Χρησιμοποιείται μέσα σε μια μέθοδο για να αναφερθούμε στο αντικείμενο για το οποίο καλείται η μέθοδος.
- Το **this** είναι μια αναφορά στο αντικείμενο στο οποίο βρισκόμαστε.
- Αν για παράδειγμα μια μέθοδος της τάξης Human έπρεπε να επιστρέφει το ίδιο το αντικείμενο:

```
Human increaseAge(){
        age++;
        return this;
}
```

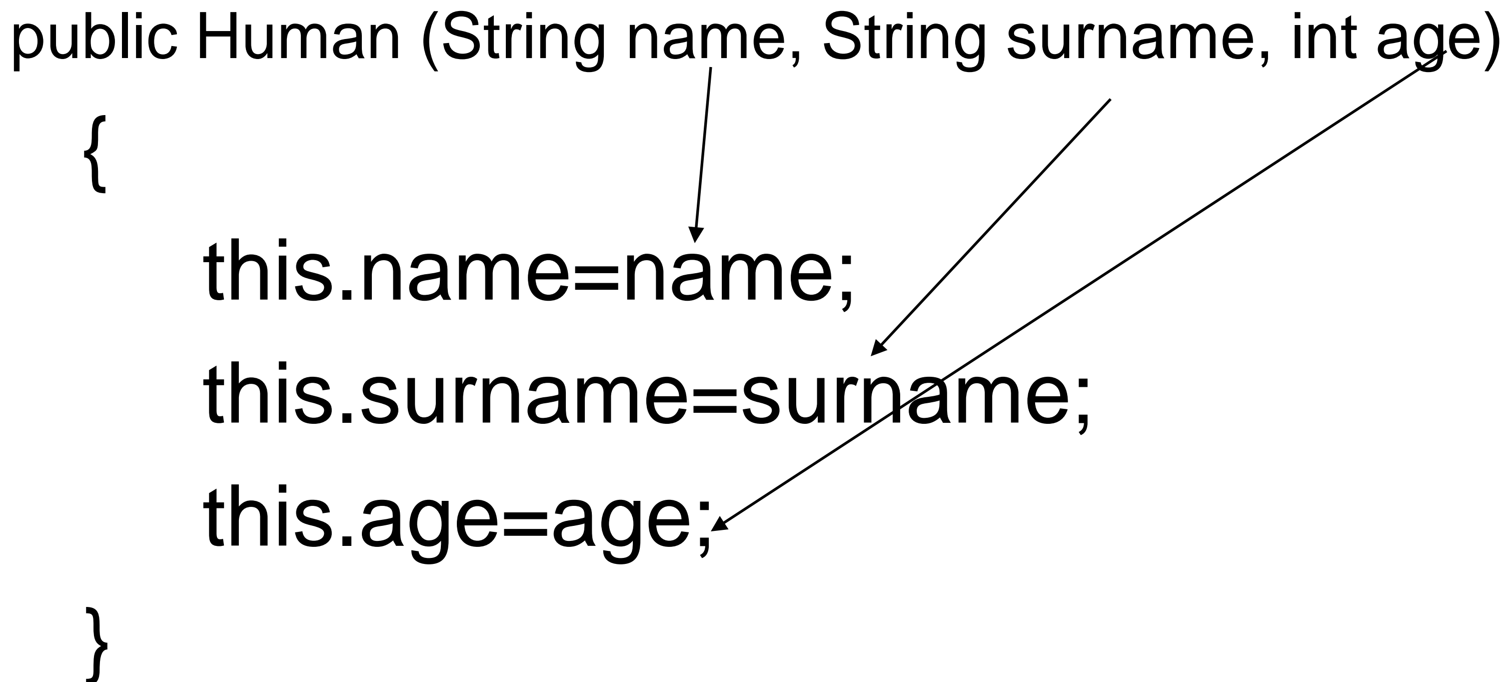
- Human h1=new Human(“Nikos”, “Nikolaou”,20);
- int x=h1.increaseAge().increaseAge().getAge();

Το **this** και οι κατασκευαστές

- Με τη χρήση του this μπορεί να καλεί ο ένας κατασκευαστής τον άλλο.

```
public Human (String tempName, String tempSurname)
{
      this (tempName, tempSurname,0);
}
```

Το **this** και οι συνωνυμίες μεταβλητών

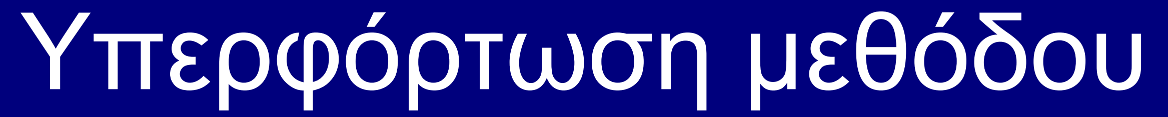


Η μέθοδος finalize()

- Η Java διαθέτει μηχανισμό απελευθέρωσης της μνήμης που δεσμεύουμε και παύουμε να χρησιμοποιούμε (garbage collector)
- Ο μηχανισμός απελευθερώνει μνήμη που δεσμεύτηκε με new (π.χ. όταν βγούμε από το μπλοκ που έγινε η δέσμευση)
- Πριν ο garbage collector αποδεσμεύσει το χώρο ενός αντικειμένου καλεί τη finalize.

Πότε ορίζουμε την finalize

- Το garbage collection δεν σημαίνει απόλυτη διαγραφή των πράξεων του αντικειμένου,
  - π.χ. αν το αντικείμενο έχει αυξήσει μια static μεταβλητή/ μετρητή θα πρέπει να τη μειώσουμε στη finalize()
- Το garbage collection δεν γίνεται άμεσα,
  - π.χ. συμβαίνει όταν χρειαστεί μνήμη το πρόγραμμα
- Το garbage collection απλά αποδεσμεύει μνήμη
  - π.χ. αν θέλουμε να παρακολουθούμε πότε αποδεσμεύεται μνήμη μπορούμε να ορίσουμε τη finalize() να τυπώνει κάποιο μήνυμα


Υπερφόρτωση μεθόδου

Υπερφόρτωση μεθόδων

- Σε μια τάξη μπορούμε να ξαναχρησιμοποιήσουμε το ίδιο όνομα για μεθόδους που έχουν ελαφρώς διαφορετική συμπεριφορά
  - Διαφορετικούς τύπους ορισμάτων
  - Διαφορετικό πλήθος ορισμάτων

```
void speak (String phrase);
void speak ();
void speak (int times, String phrase);
```

Περισσότερη υπερφόρτωση

- Δεν μπορούμε να χρησιμοποιήσουμε υπερφόρτωση μόνο στην επιστρεφόμενη τιμή

  boolean speak(String phrase); void speak (String phrase);

  Δεν είναι φανερό ποια συνάρτηση από τις δύο θα χρησιμοποιηθεί
- Γιατί να προσθέσουμε υπερφόρτωση;

  οικονομία σε ονόματα

  ευανάγνωστος κώδικας

  βιβλιοθήκες, λιγότερα ονόματα για να μάθει κανείς
- Μειονεκτήματα

  Γίνονται λάθη ευκολότερα

  μερικές φορές εμφανίζονται παραπλανητικά μηνύματα λάθους

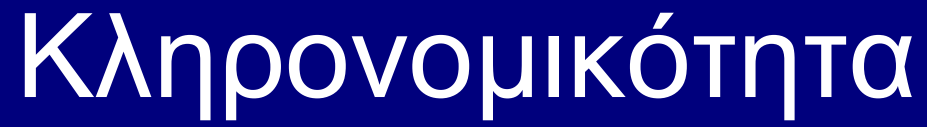
Κλήση υπερφορτωμένης μεθόδου

- Ο compiler ψάχνει αυτήν που ταιριάζει καλύτερα
  - Προτιμά ακριβές ταίριασμα από όλα τα άλλα
  - Βρίσκει την πιο κοντινή προσέγγιση
  - Επιδιώκει μόνο τις μετατροπές που έχουμε διεύρυνση τύπων και όχι στένεμα π.χ.
  - void add (int I, int j);
  - void add (double d, double e);
  - H add(10,8) θα χρησιμοποιήσει το (int, int)
  - H add(3.5, 4) θα χρησιμοποιήσει το (double, double)
- Μπορούμε να χρησιμοποιήσουμε casting για να επιβάλλουμε μια επιλογή, ή για να αποφύγουμε σύγχυση

Παράδειγμα

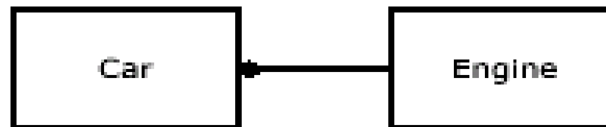
```
class Calculator{
    static int add(int a, int b) {return a+b;}
    static float add(float a, float b) {return a+b;}
    static double add(double a, double b) {return a+b;}

    public static void main(String args[] ){
        int x=5; int y=6;
        double k=5.3; double m=4.5;
        System.out.println("Atrhoisma "+ add(x,y));
        System.out.println("Athroisma "+ add(k,m));
        System.out.println("Athroisma "+ add(x,m));
    }
}
```


Κληρονομικότητα

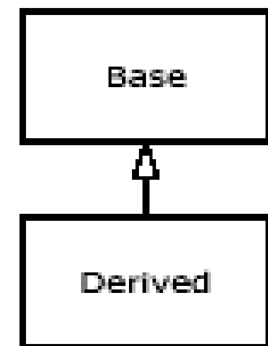
Επαναχρησιμοποίηση κλάσεων

- Δημιουργία ενός αντικειμένου μιας κλάσης
- Χρήση μιας κλάσης στον ορισμό μιας άλλης κλάσης - Σύνθεση (composition ή aggregation)
  - π.χ. Φτιάχνουμε μια κλάση«μηχανή» και στη συνέχεια μια κλάση«αυτοκίνητο» που «έχει» μία «μηχανή»



Κληρονομικότητα

- Αντιγραφή της δομής της κλάσης και επέκτασή της με νέα χαρακτηριστικά και λειτουργίες.
- Αν αλλάξει η βασική κλάση (base ή super ή parent class), τότε αλλάζει και η παράγωγη κλάση (derived ή inherited ή sub ή child class).
- Στη Java κάθε τάξη μπορεί να κληρονομεί μόνο μία κλάση

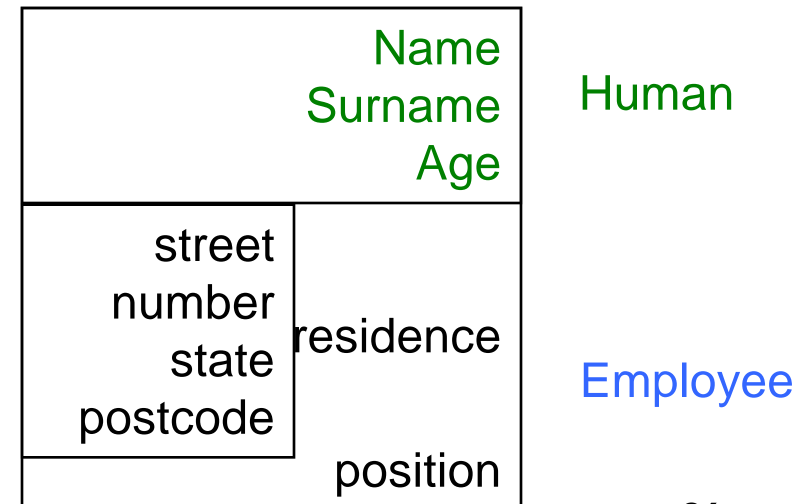


Παράδειγμα

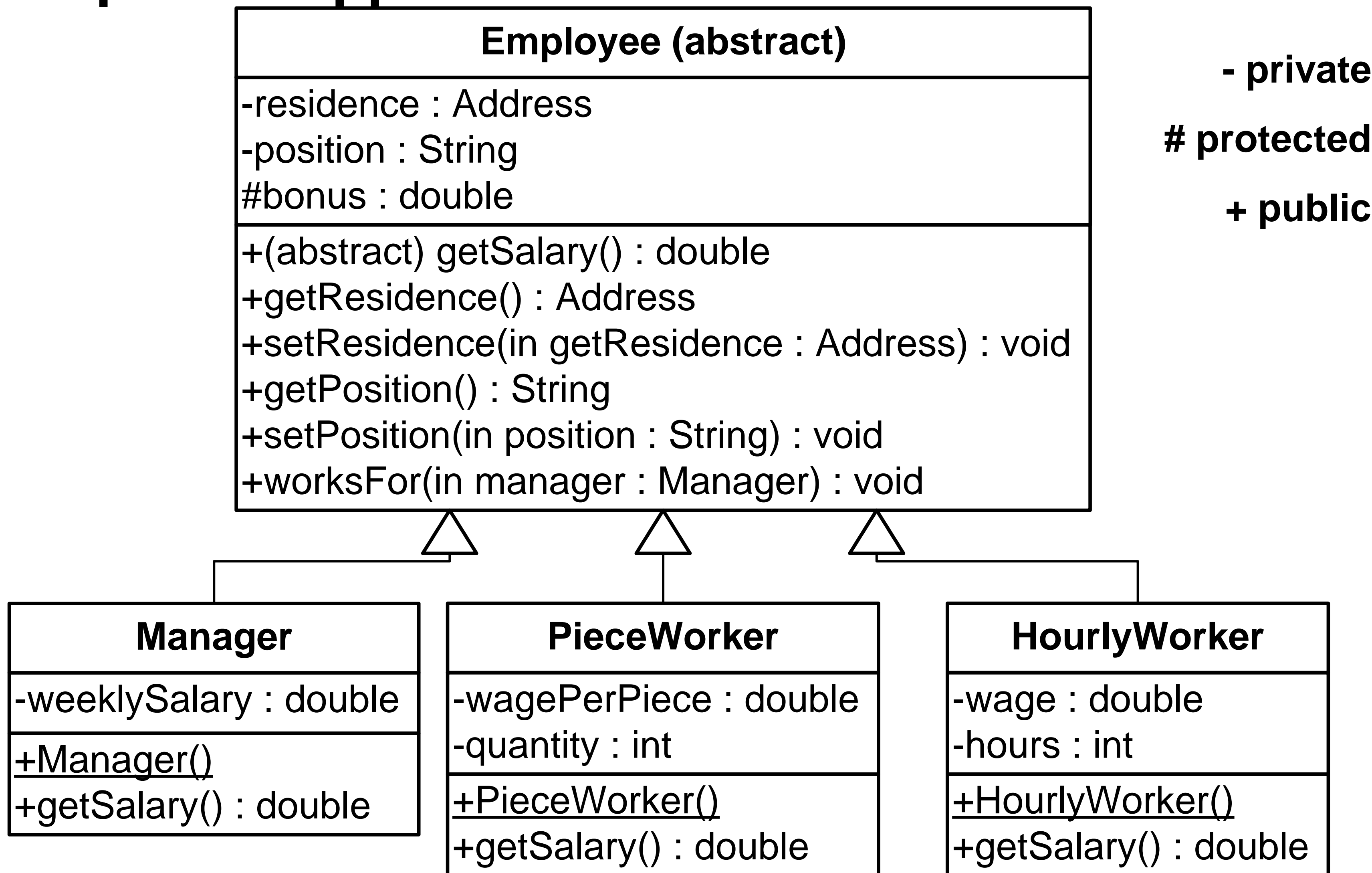
```
class Human{
    String name;
    String surname;
    int age;
    void setName(String tempName){name=tempName;}
    String getName(){return name;}
    …
}
class Address{
    String street;
    int number;
    String state;
    long postcode;
}
```

Παράδειγμα

```
public class Employee extends Human{ //κληρονομικότητα
    String position;
    Address residence;  //σύνθεση
    void setPosition(String temp){position=temp;}
    String getPosition () {return position;}
    void setAddress(Address tempad){residence=tempad;}
    Address getAddress() {return residence;}
}
… main(..){
Employee e1=new Employee();
Address a1=new Address();
a1.street="Patision"; …
e1.setAddress(a1);
}
```



Παράδειγμα



- private

 # protected

+ public

Κληρονομικότητα και ορατότητα

- Από τα μέλη της Employee:
  - Στην Employee και τα αντικείμενά της: είναι όλα ορατά private, protected και public
  - Στις Manager, PieceWorker, HourlyWorker και τα αντικείμενά τους: είναι μόνο τα public και protected (π.χ. το bonus και όλες οι μέθοδοί της) π.χ. στην Manager:

```
double getSalary(){ return weeklySalary*4+bonus;}
public Manager(){
        position="manager"; //ΛΑΘΟΣ: το position
  είναι protected
        setPosition("manager");  //ΣΩΣΤΟ
}
```

  - Σε οποιαδήποτε άλλη κλάση: είναι μόνο τα public (π.χ. οι μέθοδοι)

Κληρονομικότητα και κατασκευαστές

- Οι κατασκευαστές δεν κληρονομούνται
  - Είναι στενά δεμένοι στην κλάση με τον ορισμό τους
- Αν η Employee όριζε δύο κατασκευαστές

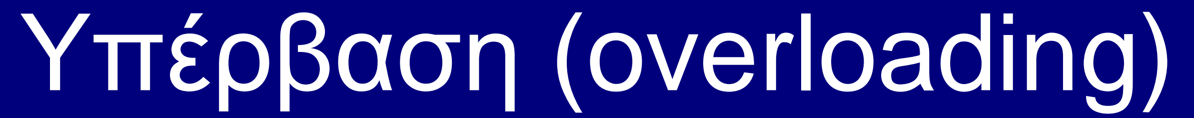
```
public Employee(String nm, int wage, int hours,
    double attitude)
public Employee(String nm, int wage)
```

- και η Manager όριζε ένα constructor

```
public Manager(String nm, int wage, int hours, double
    attitude, Employee under)
```

- Τότε η Manager δεν μπορεί να κληρονομήσει τους κατασκευαστές των 2 και 4 ορισμάτων

```
Manager b = new Manager("Ralph", 25); // Λάθος
Manager s = new Manager("Pat",25,50,.8,null); // Σωστό
```


Υπέρβαση (overloading)

Υπέρβαση μεθόδων

- Η getCompleteData της Human

```
String getCompleteData(){
   String data=new String();
   data=name+“ “+surname+” “+age+” years old”;
   return data;
}
```

- Λόγω κληρονομικότητας μπορούμε να την καλέσουμε για έναν Employee, αλλά δε θα έχουμε όλες τις πληροφορίες γι’αυτόν.

Η λύση

- Υπερβαίνουμε τη μέθοδο, δηλώνοντάς την ***ΚΑΙ*** στην Employee με το ίδιο όνομα
- Καλούμε στο σώμα της τη μέθοδο της employee με τη δήλωση ***super***

```
class Employee{
...
String getCompleteData(){
    String data=new String();
    data=super.getCompleteData() + " "+
        residence.getCompleteAddress()+" position:"+
        position;
    return data;
}
}
```

Υπέρβαση της toString

- Η τάξη Object την οποία κληρονομεί εξ ορισμού κάθε τάξη στη Java έχει μια ***public*** μέθοδο toString η οποία επιστρέφει String.
- Μπορούμε να την υπερβούμε και στις τρεις τάξεις που φτιάξαμε μετονομάζοντας τις getCompleteData και getCompleteAddress.
- Πρέπει υποχρεωτικά να είναι public γιατί στην Object είναι public.
- Αλλάζει ο τρόπος κλήσης της, π.χ. στην Employee

```
public String toString(){
  String data=new String();
  data=super.toString() + " "+residence+" position:"+position;
  return data;
}
```

Τελικοί (Final) μέθοδοι

- Οι “final” μεταβλητές γίνονται σταθερές
  - Ανατίθεται ακριβώς μία φορά και δε μπορεί να αλλάξει
- Οι “final” μέθοδοι δεν είναι overridable
  - Η κλάση γονέα τις ορίζει μία φορά και δεν ορίζονται ξανά στις υπο-κλάσεις
  - Όλες οι private μέθοδοι είναι έμμεσα τελικές (implicitly final)
  - Εξασφαλίζουμε ότι η συμπεριφορά διατηρείται και δεν μπορεί να τις αλλάξει κανείς στις υπο-κλάσεις

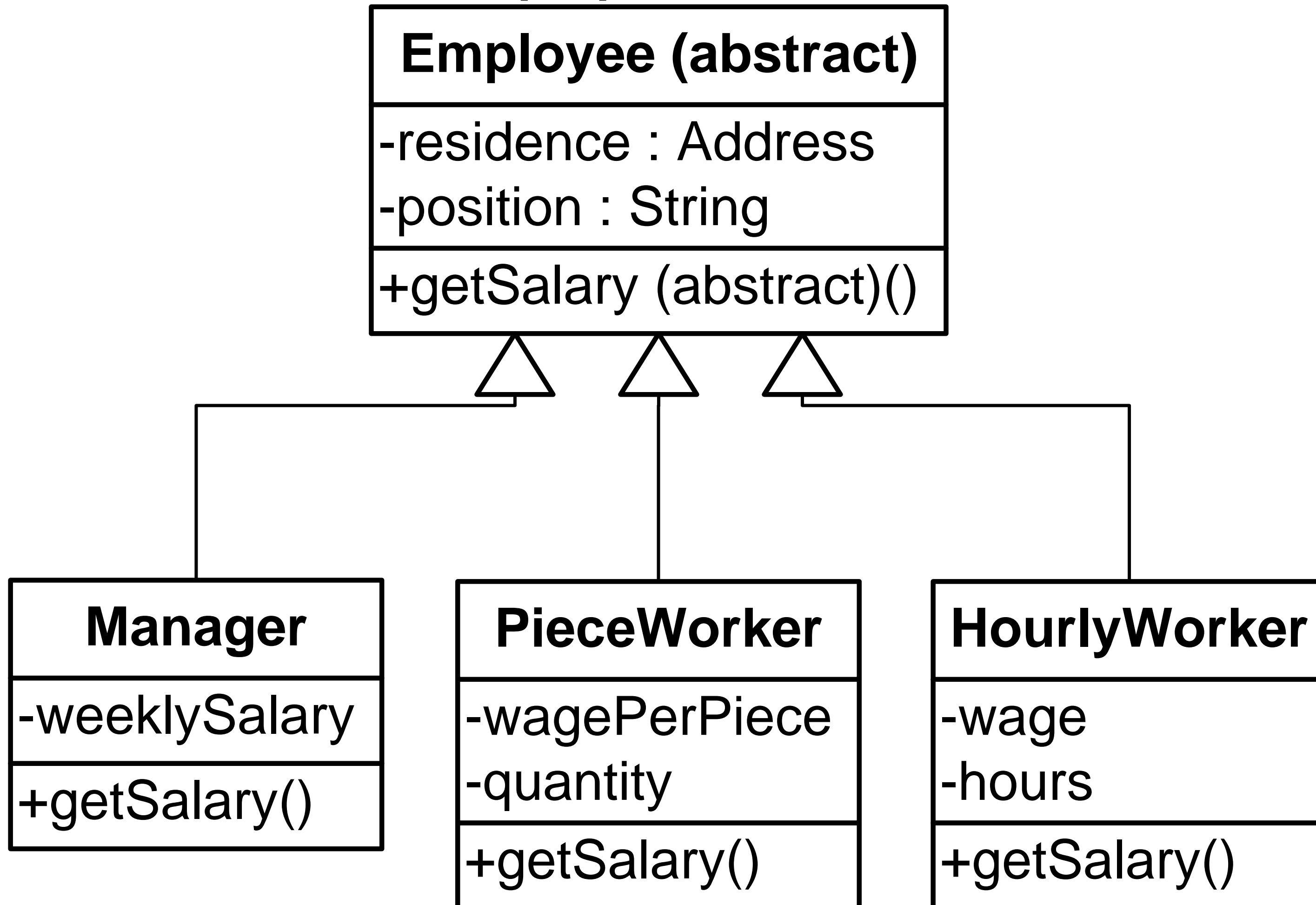
Final κλάσεις

- Οι “final” κλάσεις δεν κληρονομούνται
  - Ολοι οι μέθοδοι της γίνονται έμμεσα τελικές
  - Οταν θέλουμε να είμαστε σίγουροι ότι κανείς δεν θα τις κληρονομήσει

    public final class String{ …..}

- Οι final μέθοδοι και κλάσεις δεν χρησιμοποιούνται συχνά

Παράδειγμα (1)



Παράδειγμα (2)

```
public abstract class Employee {
    ...
    public abstract double getWeeklySalary(); // ορίζεται στις απόγονες
}
public final class Manager extends Employee {
    private double weeklySalary;
    public double getWeeklySalary( ) {return weeklySalary;}
}
public final class PieceWorker extends Employee {
    private double wagePerPiece; // μισθός ανά τεμάχιο
    private int quantity; //τεμάχια παραγωγής
    public double getWeeklySalary( ) {return wagePerPiece*quantity;}
}
public final class HourlyWorker extends Employee {
    private double wage; // μισθός ανά ώρα
    private double hours; //ώρες εργασίας
    public double getWeeklySalary( ) {return wage*hours;}
}
```

Διαχείριση
αντικειμένων

Κατασκευαστές

```
public Human(){
    name=“Unknown”; surname=“Unknown”; age=0;
    System.out.println(“A new Human has been created”);
}

public Address(){
    street=“Unknown”; number=0; city=“Unknown”;
    System.out.println(“A blank Address has been created”);
}

public Employee(){
    residence=new Address();
    position=“Unemployed”;
    System.out.println(“A new Employee has been created”);
}
```

- Με ποια σειρά καλούνται οι κατασκευαστές;

Δημιουργία αντικειμένων

Address ad1=new Address();

```
A blank Address has been created
```

Human h1=new Human();

```
A new Human has been created
```

Employee e1= new Employee();

```
A new Human has been created
A blank Address has been created
A new Employee has been created
```

- Καλείται αυτόματα ο βασικός κατασκευαστής της Human

Αν η Human δεν έχει βασικό κατασκευαστή;

```
public Human(String name,String surname, int age){
    this.name=name; this.surname=surname; this.age=age;
    System.out.println(name+” “+surname+“ has been created”);
}
```

- Πρέπει να δηλώσουμε στον κατασκευαστή της Employee ποιο συγκεκριμένο κατασκευαστή της Human θα καλέσει
- Αυτό γίνεται με τη λέξη ***super***.

```
public Employee(){
    super(“Unknown”,”Unknown”,0);
    residence=new Address();
    position=“Unemployed”;
    System.out.println(“A new Employee has been created”);
}
```

Παράδειγμα χρήσης

```
class Demo {
    public static void main (String args[]){
      Employee emp1=new Employee();
    }
}
```

```
Unknown Unknown has been created
A blank Address has been created
A new Employee has been created
```

Καταστροφείς - finalize

```
class Human{…
protected void finalize(){
  System.out.println("The human has been cleared");
}…}

class Employee{…
protected void finalize(){
   super.finalize();
   System.out.println("The employee has been cleared");
}…}
```

Με ποια σειρά καλούνται οι καταστροφείς;

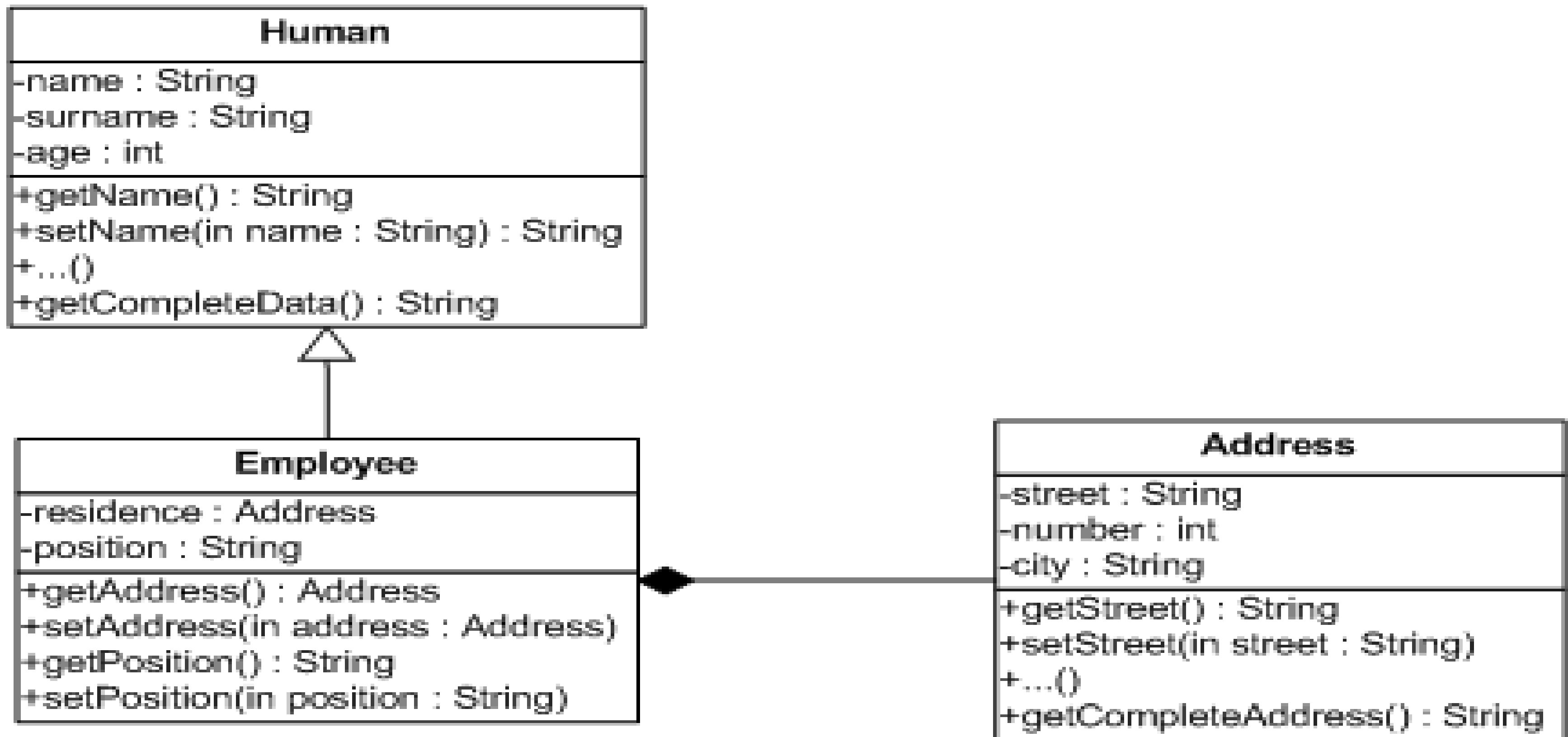
```
class Demo {
public static void main (String args[]){
    new Employee(); //φτιάχνει ένα αντικείμενο άχρηστο
    System.gc();  //καλεί τον garbage collector
}}
```

```
Unknown Unknown has been created
A new Address has been created
A new employee has been created
The human has been cleared
The employee has been cleared
The address has been cleared
```

- Αφού καταστραφεί ο employee είναι άχρηστο το αντικείμενο address

Πολυμορφισμός

Κληρονομικότητα - Παράδειγμα



Δομές και κληρονομικότητα

- Σε ένα array με αντικείμενα Human μπορούμε να βάλουμε και αντικείμενα Employee (***upcasting***)

```
Human[] group=new Human[];
group[0]=new Human();
group[1]=new Employee();
```

- Στα αντικείμενα αυτά μπορούμε χωρίς κίνδυνο να καλέσουμε χαρακτηριστικά και μεθόδους της Human.
- Για να καλέσουμε χαρακτηριστικά και μεθόδους της Employee από κάποιο αντικείμενο πρέπει πρώτα να το μετατρέψουμε σε Employee (***downcasting***)

```
(Employee)group[1].getPosition();
(Employee)group[0].getPosition(); //Class Cast Exception
```

Η λύση - RTTI

- Για τη μέθοδο toString που υπάρχει και στις δύο τάξεις, το πρόβλημα λύνεται αυτόματα.
- Χωρίς να κάνουμε downcasting.

```
group[1].toString();
group[0].toString();
```

- Αν βρει αντικείμενο της τάξης Human καλεί την toString της Human. Αν βρει αντικείμενο της τάξης Employee καλεί αυτόματα την αντίστοιχη toString.
- Run Time Type Identification – Καθορισμός τύπου την ώρα εκτέλεσης

Δομές αντικειμένων

- Σε ArrayList και Vector αποθηκεύουμε αντικείμενα διαφόρων τάξεων που όλες κληρονομούν από την ίδια βασική τάξη.
- Η βασική τάξη έχει μεθόδους και οι παράγωγες τάξεις τις υπερβαίνουν
- Όταν ανακτούμε ένα αντικείμενο από τη δομή το μετατρέπουμε στη βασική τάξη και καλούμε τις μεθόδους του.
- Ανάλογα με τον τύπο του αντικειμένου παίρνουμε και άλλη συμπεριφορά - ***Πολυμορφισμός***

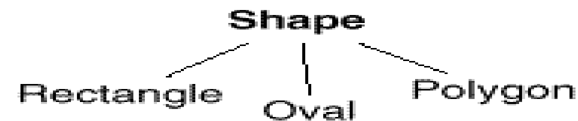
Πλεονεκτήματα του πολυμορφισμού

- Μπορούμε να ασχολούμαστε με τη γενική συμπεριφορά των αντικειμένων και να αφήνουμε τη συγκεκριμένη συμπεριφορά του καθενός να ορίζεται την ώρα εκτέλεσης
- Διευκολύνει την επέκταση. Καθώς τα μηνύματα κλήσης είναι ίδια (προς τη βασική τάξη) νέες τάξεις μπορούν να δημιουργηθούν αρκεί να καθορίσουν το δικό τους τρόπο χειρισμού των μηνυμάτων.

Abstract classes

κλάσεις αφηρημένου τύπου (abstract)

- Οι αφηρημένες κλάσεις σχεδιάζονται για να οργανώσουμε μία κοινή ιδιότητα, οι οποία όμως δεν χρησιμοποιείται από μόνη της
- π.χ. Τετράπλευρο, Έλλειψη, Πολύγωνο ..
  - έχουν πολλά κοινά



  - Θέλουμε να οργανώσουμε αυτή την κοινή συμπεριφορά σε έναν γονέα γενικού τύπου
  - με το όνομα “Shape"

Αφηρημένες τάξεις – abstract classes

- Μια abstract τάξη βρίσκεται στην κορυφή μιας ιεραρχίας τάξεων και συγκεντρώνει λειτουργίες.
- Οι υπόλοιπες τάξεις της ιεραρχίας υλοποιούν τις λειτουργίες αυτές με το δικό τους τρόπο
- Δεν μπορούμε να φτιάξουμε αντικείμενα abstract τάξεων μπορούμε όμως να έχουμε αναφορές σε abstract τάξεις.

Interfaces

Abstract class- Interface

- Για μια κλάση που δηλώνεται abstract δεν μπορούμε να φτιάξουμε αντικείμενα.
- Μπορούμε να δηλώσουμε κάποια λειτουργικότητα και κάποια βασικά χαρακτηριστικά που θα τα κληρονομήσουν οι απόγονες κλάσεις.
- Τις abstract κλάσεις που ορίζουν μόνο μεθόδους τις δηλώνουμε ως διεπαφές – interfaces
- Τα interfaces συγκεντρώνουν μόνο ***δηλώσεις λειτουργικότητας.*** Άλλες κλάσεις αναλαμβάνουν να υλοποιήσουν (***implement***) τις δηλώσεις αυτές

Interface

- Το interface είναι μια συλλογή από “υπογραφές“ μεθόδων (δεν υπάρχουν στιγμιότυπα, ούτε υλοποιήσεις των μεθόδων)
- Περιγράφει πρωτόκολλο/συμπεριφορά αλλά όχι υλοποίηση
- Όλες οι μέθοδοι του είναι public και abstract (ποτέ static)
- Όλες οι μεταβλητές είναι static και final
- Μια κλάση υλοποιεί (implements) ένα interface

Πλεονεκτήματα

- Δηλώνουν μια επιθυμητή λειτουργικότητα και αφήνουν στις τάξεις να την ορίσουν

```
public interface Shape {
    public abstract double area(); // calculate area
    public abstract double volume(); // calculate volume
    public abstract String getName()// return shape name
}
public class Triangle implements Shape {…}
```

- Υποχρεωτικά θα πρέπει να ορίσει τις μεθόδους της διεπαφής Shape
- Είναι ένας έμμεσος τρόπος να έχουμε πολλαπλή κληρονομικότητα λειτουργιών στη Java

Παράδειγμα – Enumeration, Iterator

- java.util.Enumeration: είναι ένα interface. Περιγράφει μεθόδους για το ψάξιμο μέσα σε μια συλλογή

```
public interface Enumeration{
  boolean hasMoreElements();
  Object nextElement();  }
```

- Ένας iterator υλοποιεί αυτό το interface
- Οι κλάσεις Vector, Hashtable, Set, Graph, Tree κλπ. υλοποιούν το Enumeration ορίζοντας πώς θα ανταποκρίνεται η κάθε μέθοδος
- Μπορούμε να χρησιμοποιήσουμε το interface ως όνομα τύπου σε όσες κλάσεις υλοποιούν το interface.

```
void printAll(Enumeration e) {
  while(e.hasMoreElements())
    System.out.println(e.nextElement());}
```

Παράδειγμα: VectorEnumerator

```
class VectorEnumerator implements Enumeration{
        private Vector vector;
        private int count;
        VectorEnumerator(Vector v) {
                vector = v;
                count = 0;  }
        public boolean hasMoreElemets() {
                return count < vector.size();  }
        public Object nextElement() {
                return vector.elementAt(count++);}
    }
```

Σύνταξη του interface

- ΟΛΕΣ οι μέθοδοι ενός interface πρέπει να υλοποιηθούν, αλλιώς η νέα κλάση θα πρέπει να οριστεί ως abstract
- Οι μέθοδοι του interface πρέπει να είναι public
- Μια κλάση μπορεί να υλοποιήσει πολλαπλά interfaces

  public class Shape implements Colorable, Printable { …}

- Πρέπει να προσέχουμε τα ονόματα των μεθόδων στα interfaces που θα συνδυαστούν να μη συμπίπτουν γιατί δημιουργούνται συγχύσεις.

Κληρονομικότητα Interfaces

- Τα interfaces μπορούν να επεκτείνουν αλλά interfaces

```
public interface Beepable {
    public void beep();  }
  public interface VolumeControlled extends Beepable {
    public int getVolume (int newVol);
    public void setVolume( int newVol);
    public void mute();  }
```

- Η κλάση είναι συμβατή με τον τύπο του interface. Κάνουμε upcasting σε τύπο interface όπως θα κάναμε σε μια abstract ή σε μια οποιαδήποτε κλάση

Interfaces ή abstract κλάσεις ?

- Παρόμοιες χρήσεις
  - Σχεδιάστηκαν για την ομαδοποίηση της συμπεριφοράς,
  - για να επιτρέπουν το upcasting,
  - για την εκμετάλλευση του πολυμορφισμού
- Μια κλάση μπορεί να υλοποιήσει πολλαπλά interfaces, αλλά έχει μόνο μια υπερ-κλάση
- Το interface δεν έχει καθόλου υλοποίηση
  - Είναι καλό, αν η ομοιότητα συμπεριφοράς είναι μόνο στο όνομα
  - Είναι κακό, αν υπάρχει κοινός κώδικας που θα μπορούσε να μεταφερθεί στην abstract κλάση (σε μια όχι abstract μέθοδο)
- Αν υπάρχει κοινός κώδικας➔abstract κλάση, αλλιώς interface

Τα πεδία των interfaces

- Είναι static και final και μπορούν να χρησιμοποιηθούν για να ομαδοποιήσουν σταθερές

  public interface Months {

  int JANUARY = 1, ..., DECEMBER = 12; }

  ...

  Από κάθε άλλη κλάση: Months.JANUARY

- Αρχικοποίηση γίνεται όταν αναφερθούμε για πρώτη φορά στο Interface.

  public interface RandVals {

  *int rint = (int)(Math.random() * 10);}*

  …

  RandVals.rint;

Εσωτερικά interfaces και κλάσεις

- Μέσα σε ένα interface (σε μία κλάση) μπορούμε να δηλώσουμε ένα άλλο interface (μια κλάση)
- Χρησιμοποιούνται
  - για να ομαδοποιήσουμε σχετικά μεταξύ τους interfaces ή κλάσεις (λειτουργικότητα), που δε χρησιμοποιούνται σε άλλο περιεχόμενο.
  - για να διαχωρίσουμε τη λειτουργικότητα σε μια κλάση από την ίδια την κλάση
- Παράδειγμα:
  - μια κλάση BinaryTree μπορεί να έχει τις μεθόδους για προσθήκη και αφαίρεση κόμβων.
  - Μέθοδοι που υλοποιούν κάποιο αλγόριθμο ταξινόμησης ή αναζήτησης κόμβων ομαδοποιούνται σε εσωτερική κλάση της BinaryTree. Διαχωρίζοντας έτσι τις λειτουργίες