



# Προγραμματισμός II (Java)

## 3. Μέθοδοι

# Περιεχόμενα

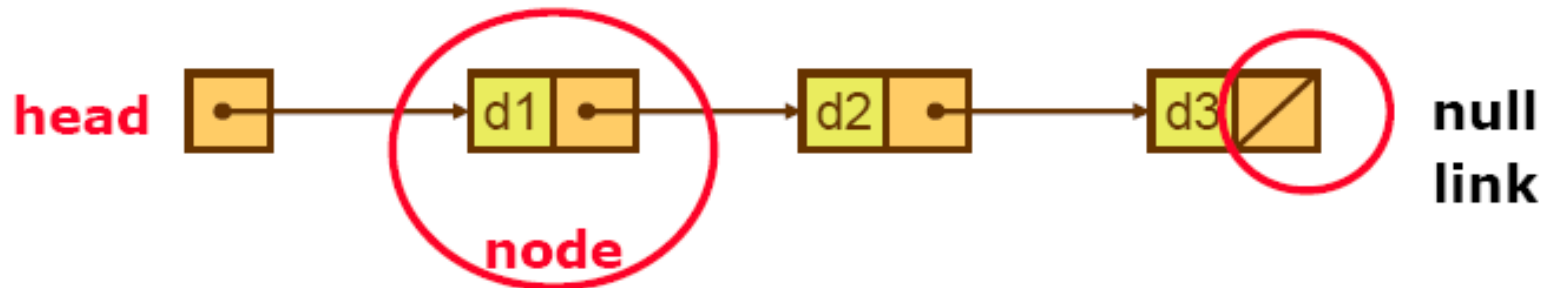
- Δυναμικές δομές
  - Vector
  - ArrayList
- Πολυμορφισμός - Polymorphism
- Αφηρημένες κλάσεις – Abstract Classes
- Διεπαφές - Interfaces

# Περιορισμοί των πινάκων

- Οι εισαγωγές και οι διαγραφές σε κάποια θέση «κοστίζουν»
  - Πρέπει να μετακινηθούν όλα τα υπόλοιπα στοιχεία για να καλύψουν το κενό ή να δημιουργήσουν χώρο.
  - Αν γεμίσουν οι θέσεις θα πρέπει να αντιγράψουμε όλα τα στοιχεία σε νέο, μεγαλύτερο πίνακα.
- Η λογική διαδοχή των θέσεων του πίνακα ταυτίζεται με την πραγματική (ο πίνακας αποθηκεύεται σε συνεχόμενες θέσεις μνήμης)
- Θέλουμε να αποσυνδέσουμε τη λογική από την πραγματική διαδοχή θέσεων.
  - Έτσι θα έχουμε διαγραφές και εισαγωγές χωρίς μετακίνηση

# Δυναμικές δομές δεδομένων

- Περιέχουν **αντικείμενα οποιουδήποτε τύπου**
- Η κλάση Vector
  - Επιτρέπει να φτιάξουμε δομές όπως οι πίνακες που το μέγεθός τους αυξομειώνεται δυναμικά.
- Η κλάση ArrayList
  - Υλοποιεί τη δομή της λίστας στη Java
  - Κάθε στοιχείο (κόμβος) αποτελείται από το αντικείμενο και



# Η κλάση Vector

- Δεσμεύουμε ένα αρχικό μέγεθος και στη συνέχεια αυτό αυξάνεται όποτε χρειαστεί.
- Αν δεν καθορίσουμε τον τύπο των στοιχείων που αποθηκεύει τότε τα πάντα αποθηκεύονται και εξάγονται ως “Object”
  - `Vector group = new Vector();`
- Διαφορετικά:
  - `Vector<Human> children = new Vector<Human>();`
- Προσθέτουμε στοιχεία
  - `children.add(“Hello”);` //στο τέλος
  - `children.insertElementAt(new Employee(),0);` //σε θέση
- Αντικαθιστούμε στοιχεία
  - `children.setElementAt(new Employee(),0);` //σε θέση
- Διαγράφουμε στοιχεία
  - `boolean deleted = children.removeElement(x)` //το πρώτο x
  - `children.removeElementAt(1)` //το 2<sup>ο</sup> στοιχείο αν υπάρχει
  - `children.removeAllElements()` //όλα τα στοιχεία

# Άλλες μέθοδοι

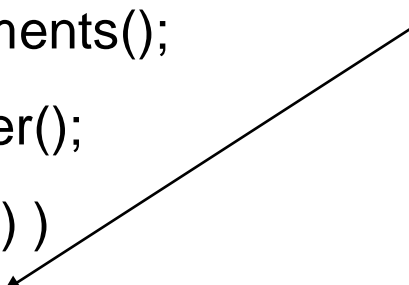
- `firstElement(), lastElement()`
  - Επιστρέφει το πρώτο/τελευταίο `Object` στο `Vector`
  - Χρειάζεται casting
- `isEmpty()`
  - Επιστρέφει `true/false`
- `contains(Object searchkey)`
  - Χρησιμοποιεί την `equals` για την αναζήτηση
- `indexOf(Object searchkey), lastIndexOf(...)`
  - Επιστρέφει τη θέση πρώτης/τελευταίας εμφάνισης του `searchkey` στο `Vector`, αλλιώς `-1`
- `size(), capacity()`

# Απαρίθμηση στοιχείων Vector

- Η μέθοδος `elements()` επιστρέφει μια απαρίθμηση των στοιχείων του `Vector`.

Καλεί αυτόματα την μέθοδο `toString()` για κάθε αντικείμενο στο `vector`.  
Δεν έχει πάντα τα επιθυμητά αποτελέσματα

```
Enumeration enum = vector.elements();  
StringBuffer buf = new StringBuffer();  
while ( enum.hasMoreElements() )  
    buf.append( enum.nextElement() ).append( " " );  
System.out.println(buf.toString());
```



# Η κλάση ArrayList

- Ίδιες ιδιότητες με τη Vector
- Προτείνεται από τους δημιουργούς της Java
- **ArrayList list = new ArrayList();**
- add(Object searchkey), get(int position)

```
Iterator it = list.iterator();
```

```
StringBuffer buf = new StringBuffer();
```

```
while (it.hasNext())
```

```
    buf.append( it.next() ).append( " " );
```

```
System.out.println(buf.toString());
```



# Άλλες μέθοδοι της ArrayList

void **clear**(); //αδειάζει τη λίστα

Object **clone**(); //δημιουργεί αντίγραφο

boolean **addAll**(Collection c); //προσθέτει όλα τα  
//αντικείμενα της c στη λίστα

boolean **contains**(Object elem); // αναζητά το elem

int **indexOf**(Object elem); //βρίσκει την θέση 1<sup>ης</sup>  
//εμφάνισης

Object **remove**(int index); //διαγράφει στοιχείο

Object[] **toArray**(); // επιστρέφει τα στοιχεία της  
//λίστας σε πίνακα αντικειμένων

# Μειονέκτημα των containers

(Vector, ArrayList κλπ).

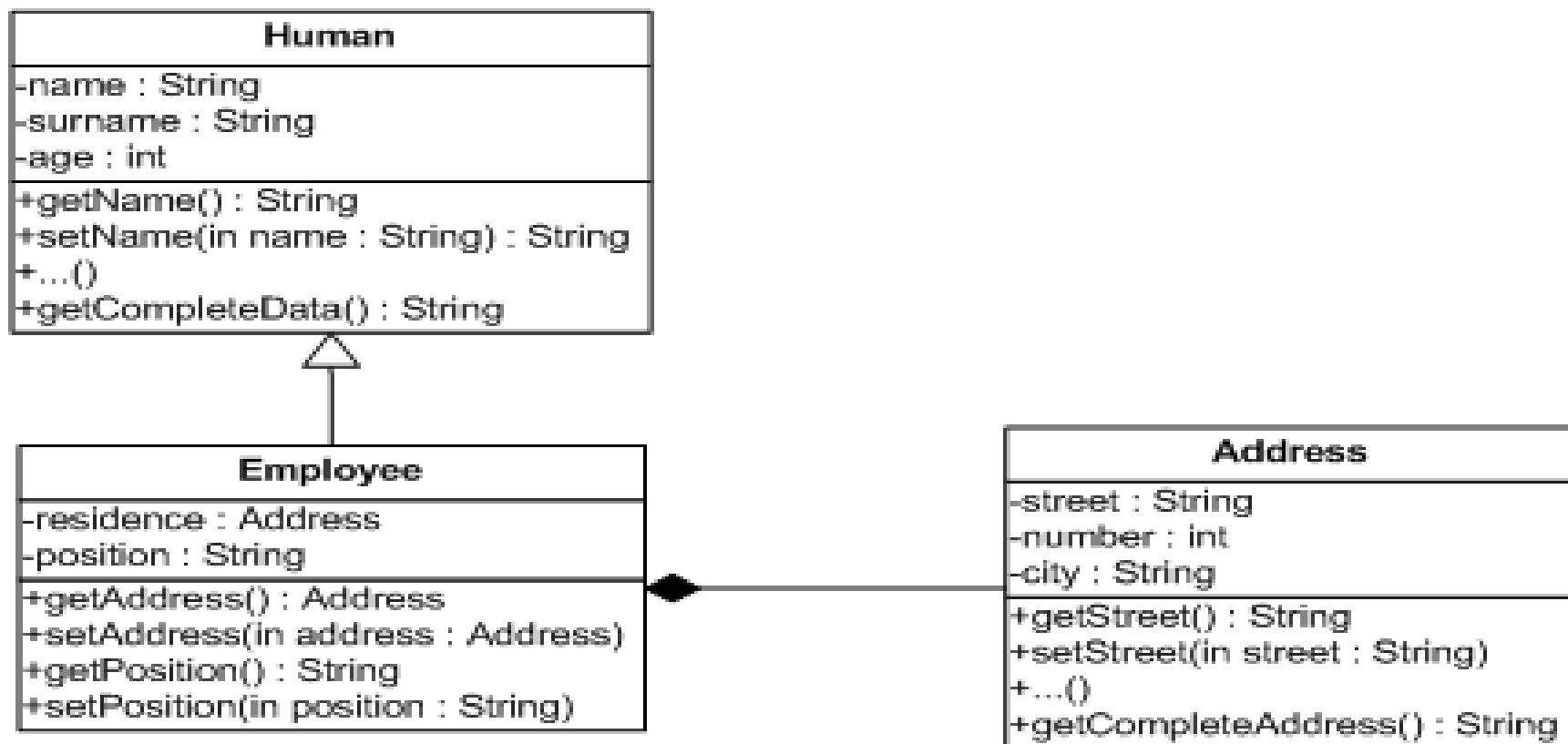
- Αν δε χρησιμοποιηθεί ο περιορισμός τύπου:
  - Όλα τα στοιχεία αποθηκεύονται ως Object
  - Πρέπει να μετατραπούν (cast) για να χρησιμοποιηθούν οι μέθοδοί τους.
- Τα containers μπορεί να περιέχουν αντικείμενα διαφόρων κλάσεων  
`list.add(new Employee());`  
`list.add(new Address());`  
`list.add(new Manager());`  
`((Employee)list.get(0)).getName();`  
`((Address)list.get(1)).getCity();`  
`((Address)list.get(2)).getCity(); //ΛΑΘΟΣ casting`

Όπως πάντα... το μειονέκτημα ...  
έγινε τελικά πλεονέκτημα



# Πολυμορφισμός

# Κληρονομικότητα - Παράδειγμα



# 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

# 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 **KAI** 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;
}
```

# 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.



# 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.

# 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)
  - Οι μέθοδοι δεν μπορούν να είναι μαζί abstract και final - **γιατί;**
  - Εξασφαλίζουμε ότι η συμπεριφορά διατηρείται και δεν μπορεί να τις αλλάξει κανείς στις υπο-κλάσεις

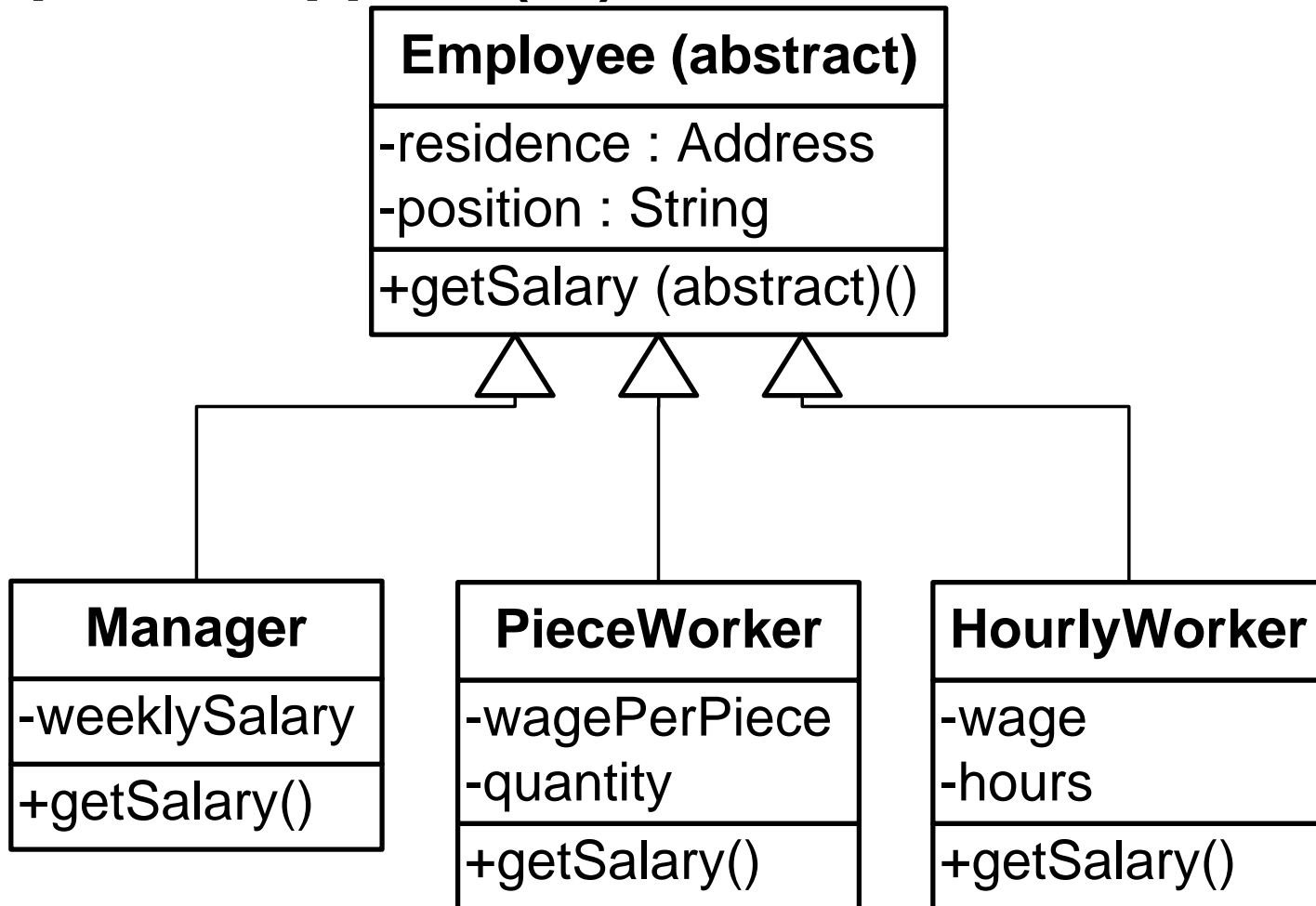
# Final κλάσεις

- Οι “final” κλάσεις δεν κληρονομούνται
  - Όλοι οι μέθοδοι της γίνονται έμμεσα τελικές
  - Όταν 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;}  
}
```



# Μελέτη περίπτωσης

- Δημιουργήστε τις τάξεις του προηγούμενου παραδείγματος
- Φτιάξτε μια δομή με το όνομα company που θα αποθηκεύει υπαλλήλους μιας εταιρίας και στο τέλος θα υπολογίζει το συνολικό μισθό όλων των υπαλλήλων.



# 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 hasMoreElements() {  
        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



# Χρήσιμες μέθοδοι

# Σύγκριση: Η μέθοδος equals

- Για να δουλέψουν οι προηγούμενοι μέθοδοι για λίστες με αντικείμενα δικών μας κλάσεων πρέπει στις κλάσεις μας να έχουμε μια μέθοδο equals π.χ.

```
public boolean equals(Object o){
    Department d=(Department)o; // πιθανό να παράγει
                                   //ClassCastException
    if (this.id==d.getId() && this.name.equals(d.getName()) &&
        this.numStudents==d.getNumStudents())
        return true;
    else
        return false;
}
```

# Ταξινόμηση

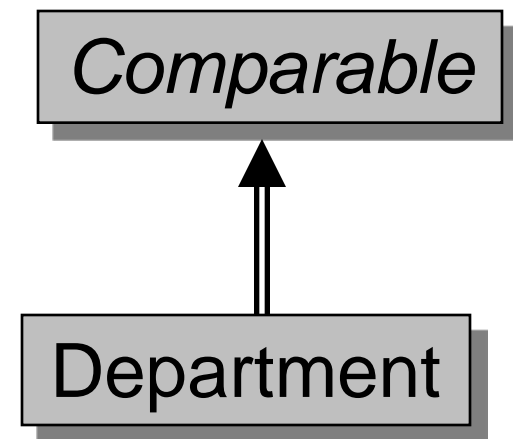
- Με ποιο τρόπο μπορώ να ταξινομήσω τα στοιχεία ενός πίνακα ή μιας λίστας; BubbleSort:


```
public void bubbleSort(int[] unsortedArray, int length) {  
    int temp, counter, index;  
    for(counter=0; counter<length-1; counter++) {  
        for(index=0; index<length-1-counter; index++) {  
            if(unsortedArray[index] > unsortedArray[index+1]) {  
                temp = unsortedArray[index];  
                unsortedArray[index] = unsortedArray[index+1];  
                unsortedArray[index+1] = temp;  
            }  
        }  
    }  
}
```

# Διάταξη

- Η διάταξη στους ακεραίους είναι δεδομένη
- Τι γίνεται όμως με τις δικές μας κλάσεις;
- Πώς μπορούμε να ορίσουμε διάταξη στα αντικείμενά τους;

```
public interface Comparable  
{  
    int compareTo(Object o);  
}
```





```
public class Department implements Comparator{
    ...
    public int compareTo(Object o){
        Department d=(Department)o; // πιθανό να παράγει
                                     //ClassCastException
        if (this.numStudents>d.getNumStudents())
            return 1;
        else if (this.numStudents<d.getNumStudents())
            return -1;
        else
            return 0;
    }
}
```



# Ταξινόμηση

```
Collections.sort(allDeps);
```

Ταξινομεί τα τμήματα με βάση τον αριθμό  
σπουδαστών που έχουν

Χρησιμοποιεί την QuickSort