Προγραμματισμός ΙΙ (Java)

Επανάληψη

Ύλη

- Οι **διαφάνειες** των διαλέξεων
- Τα παραδείγματα κώδικα στο «βοηθητικό υλικό»
- Τα κεφάλαια **8-11,14,17,20,22,25,26,28** από το βιβλίο (Deitel & Deitel 8η έκδοση)

ή

- Τα κεφάλαια **8-11, 13,14,17,19,22,23,25** από το βιβλίο (Deitel & Deitel 6η έκδοση)

Βαθμολογία

- Βαθμολογία :
  - Τελική εξέταση:70% [βαθμός >=5]
  - Εργασίες:30% [βαθμός >=5]
  - Bonus μέχρι 15%
- Τελική εξέταση: *με ανοιχτό βιβλίο και σημειώσεις μαθήματος*
  - *Ερωτήσεις πολλαπλών απαντήσεων + συγγραφή κώδικα*
- Κατοχύρωση:
  - *Εργασίες δίνετε **μόνο** μέσα στο εξάμηνο και ο βαθμός τους κατοχυρώνεται για το Σεπτέμβριο*
  - *Τίποτε δεν κατοχυρώνεται για επόμενη χρονιά!*

1ο μάθημα

- Κλάσεις
  - Δήλωση κλάσης,
  - Δήλωση χαρακτηριστικών, μεθόδων
- Αντικείμενα
  - Δημιουργία αντικειμένου
  - Κλήση μεθόδων, χαρακτηριστικών
- Έλεγχος πρόσβασης (ενθυλάκωση)

Ορισμός κλάσης

```
class Human {
        boolean alive;
        int age;
        String name;

        void born()
        {
            alive  =  true;
        }
        void speak()
        {
            System.out.println("Hi!");
        }
        void incr_age()
        {
            age = age +1;
        }
}
```

Δεδομένα

Λειτουργίες

Μέθοδοι

Στόχο έχουν να χειρίζονται τα δεδομένα, να τα τροποποιούν ή να επιστρέφουν τις τιμές αυτών, σε όποιον τις ζητήσει

Κλήση μεθόδου = Αποστολή Μηνύματος

Κλήση μεθόδων

- Μια μέθοδος καλείται από κάποιο αντικείμενο (συνήθως) π.χ.

```
Human John= new Human();
John.born();
John.speak();
John.setAge(10);
```

- Μπορούμε να αναφερθούμε απευθείας στα δεδομένα του αντικειμένου.

```
John.age=15;
int z=John.getAge();
```

Τα όρια μιας κλάσης

- Υπάρχουν τρεις τύποι πρόσβασης στα μέρη μιας κλάσης: ***public***, ***private***, και ***protected***.
- Δημόσια (***public***), τα μέρη είναι διαθέσιμα σε όλους
- Ιδιωτική (***private***), τα μέρη είναι διαθέσιμα μόνο στην ίδια την κλάση και τις λειτουργίες της
- Προστατευμένη (***protected***), τα μέρη είναι διαθέσιμα στην κλάση και σε όσες κλάσεις την κληρονομούν.
- Δεδομένη (***default***) πρόσβαση, όταν δεν δηλώνεται κάποια από τις προηγούμενες, τα μέρη είναι διαθέσιμα στην κλάση και σε όλες τις υπόλοιπες κλάσεις της ίδιας εφαρμογής (του ίδιου package)

Νέος ορισμός κλάσης

```
class Human {
        private boolean alive;
        private int age;
        void Human()
        {
            alive  =  true;
            age = 0
        }
        public int getAge()
        {
            return age;
        }
        public void setAge(int a)
        {
            age=a;
        }
        void speak()
        {
           System.out.println("Hi!");
        }
        void incr_age()
        {
           age = age +1;
        }
}
```

Συμπερασματικά

- Σε ένα πρόβλημα:
  - Προσπαθώ να δω: ποια αντικείμενα εμπλέκονται, τι δεδομένα έχει το καθένα από αυτά και ποιες λειτουργίες επιτελεί και στη συνέχεια ορίζω τις αντίστοιχες κλάσεις
- Σε μια κλάση
  - Ορίζω χαρακτηριστικά: name, surname, age
  - Ορίζω μεθόδους πρόσβασης: getName(), setName() ..
  - Ορίζω άλλες βοηθητικές μεθόδους
- Στο τέλος
  - Ορίζω μια κλάση με μια μέθοδο main()
  - Η κλάση αυτή εκτελεί το πρόγραμμά μου. Μόνο που δεν περιέχει όλη τη λύση. Κάποια πράγματα τα αναλαμβάνουν τα αντικείμενα

2ο μάθημα

- Static και Final
- Μέθοδοι
- Πέρασμα αντικειμένου σε μέθοδο
- Υπερφόρτωση μεθόδων
- Μέθοδοι κατασκευαστές
- Υπερφόρτωση κατασκευαστών
- Πακέτα
- Βοηθητικά πακέτα της Java
- Σύνθεση και κληρονομικότητα

Τελεστής ανάθεσης

- Δημιουργώντας ένα αντικείμενο, ουσιαστικά δεσμεύουμε μνήμη και δημιουργούμε μια αναφορά σε αυτή.

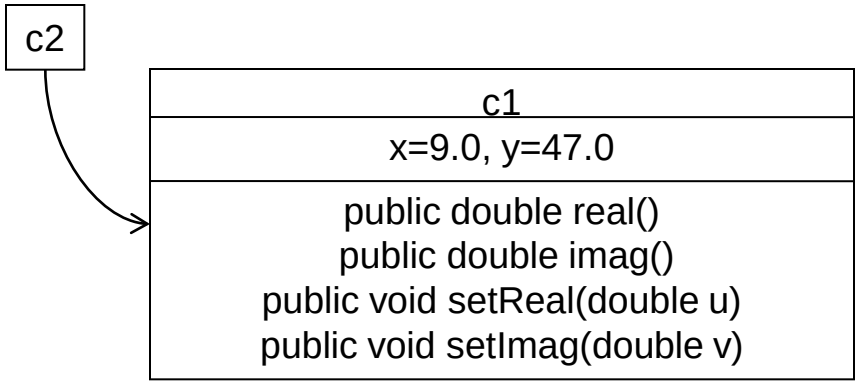
```
Complex c1 = new Complex();
Complex c2 = new Complex();
c1.setReal(9);
c1.setImag(47);
c2=c1;
```

- Όταν αναθέσουμε ένα αντικείμενο (c1) σε ένα άλλο (c2), ουσιαστικά έχουμε δύο αναφορές στην ίδια θέση μνήμης (αυτή που δεσμεύτηκε για το c1)

```
c2 = c1;
```

- Το φαινόμενο αυτό ονομάζεται aliasing (συνωνυμία)

```
public class Complex {
    private double x,y;
    public double real() { return x; }
    public double imag() { return y; }
    public void setReal(double u) {x=u; }
    public void setImag(double v) {y=v; }
}
```



Η equals στις κλάσεις χρήστη

- Ορίζοντας τη μέθοδο equals καθορίζουμε τον τρόπο με τον οποίο θα συγκρίνονται τα αντικείμενα της κλάσης μας

```
public class Complex {
   private double x,y;
    …
   public boolean equals(Complex c) {
    if (x==c.x && y==c.y)
            return true;
    else return false;
   }
}
```

static γνωρίσματα- μέθοδοι

- Αν ένα γνώρισμα είναι ***κοινό*** για όλα τα αντικείμενα μιας κλάσης τότε δεσμεύουμε μια φορά χώρο για όλα τα αντικείμενα.
- Κατά τη δήλωση της κλάσης, τη δηλώνουμε static. Το γνώρισμα αυτό θα δημιουργηθεί μία φορά μόλις δηλωθεί το πρώτο αντικείμενο αυτής της κλάσης και θα μπορεί να χρησιμοποιείται από κάθε αντικείμενο της ίδιας κλάσης.

  π.χ. static float bonus;
- Αν θέλουμε να αναφερθούμε στο γνώρισμα αυτό, μπορούμε απ' ευθείας μέσω της κλάσης:

  π.χ. Human.bonus=100,00;
- Παρόμοια ισχύουν και για τις μεθόδους.

  π.χ. static setBonus(float b)

  Human.setBonus(200,00);
- Τα static μέλη μπορούμε να τα καλούμε είτε απ'ευθείας μέσω της κλάσης είτε μέσω των αντικειμένων που δημιουργούμε. Οι static μέθοδοι έχουν πρόσβαση στα static μέλη της κλάσης.

Final γνωρίσματα

- Πρακτικά: final = μόνο για ανάγνωση
- Αν ένα γνώρισμα δηλωθεί final, τότε λειτουργεί ως σταθερά που
  - πρέπει να αρχικοποιηθεί
  - και δεν αλλάζει τιμή
- Αν η παράμετρος εισόδου μιας μεθόδου δηλωθεί final τότε δεν μπορεί να αλλάξει τιμή στο σώμα της μεθόδου.
  - public void setHour(final int hrs)
- Αν ένα αντικείμενο δηλωθεί final τότε το όνομα του αντικειμένου θα αναφέρεται πάντα στο ίδιο αντικείμενο.
- Αν μια κλάση δηλωθεί final τότε δεν μπορεί να κληρονομηθεί από άλλη κλάση.

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
  - Ολοι οι μέθοδοι της γίνονται έμμεσα τελικές
  - Οταν θέλουμε να είμαστε σίγουροι ότι κανείς δεν θα τις κληρονομήσει

    public final class String{ …..}

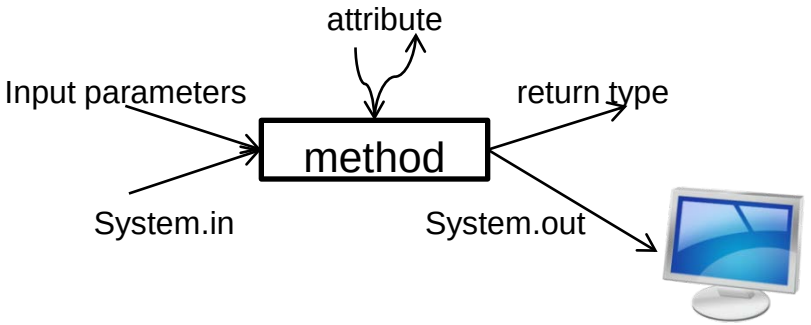
- Οι final μέθοδοι και κλάσεις δεν χρησιμοποιούνται συχνά

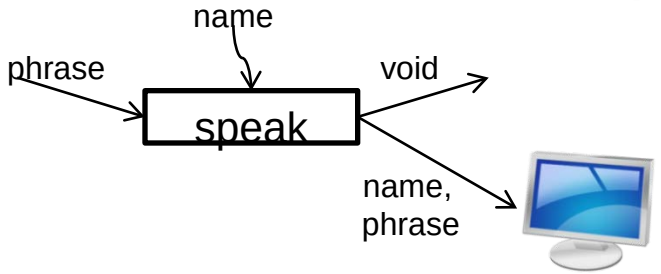
Δομή μεθόδου

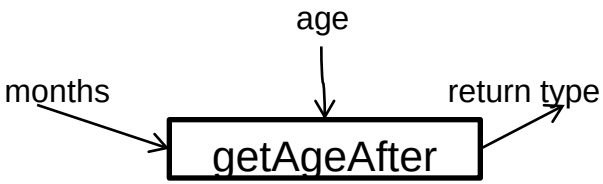
```
πρόσβαση τύποςΕπιστροφής όνομαΜεθόδου (τυπος1 παράμετρος1, …)
{
…
}

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

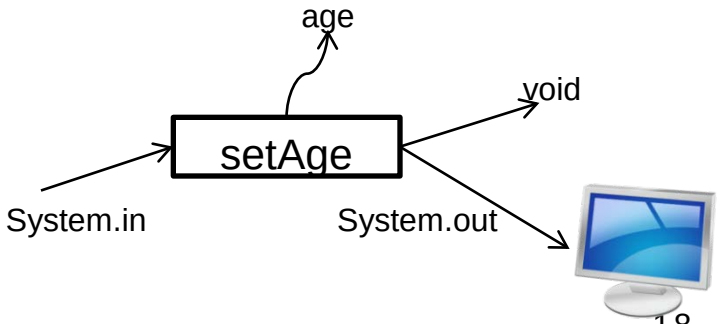
Μέθοδοι

```
class Human {
   boolean alive;
   int age;
   String name;
   void speak(String phrase)
   {
     System.out.println(“Hi!”);
     System.out.println(“My name is ”+name);
     System.out.print(“You asked me to say:”);
     System.out.println(phrase);
   }
   int getAgeAfter(int months){
        int years=months/12;
        int ageafter=age+years;
        return ageafter;
   }
   void setAge(){
        Scanner keyboard=new Scanner(System.in);
        System.out.println(“How old are you?”);
        age=keyboard.nextInt();
   }
}
```









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);
```

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**

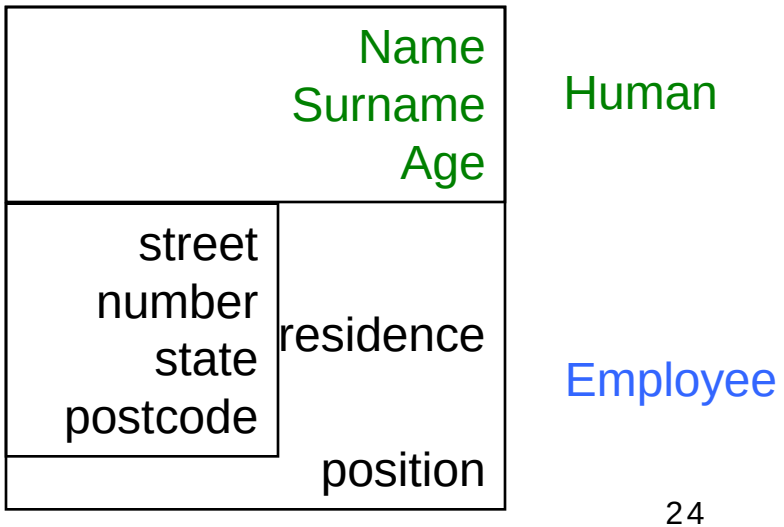
- Χρησιμοποιείται μέσα σε μια μέθοδο για να αναφερθούμε στο αντικείμενο για το οποίο καλείται η μέθοδος.
- Το **this** είναι μια αναφορά στο αντικείμενο στο οποίο βρισκόμαστε.
- Αν για παράδειγμα μια μέθοδος της τάξης Human έπρεπε να επιστρέφει το ίδιο το αντικείμενο:

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

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

Επαναχρησιμοποίηση (σύνθεση+κληρονομικότητα)

```
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);
}
```



3ο Μάθημα

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

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 1ης
//εμφάνισης
Object **remove**(int index); //διαγράφει στοιχείο
Object[] **toArray**();// επιστρέφει τα στοιχεία της
//λίστας σε πίνακα αντικειμένων

Η κλάση Arrays της java.util

- Γέμισμα πίνακα με μια τιμή
  - int[] a6=new int[5];
  - Arrays.fill(a6, 23);
- αντιγραφή
  - int[] i = new int[25];
  - int[] j = new int[25];
  - Arrays.fill(i, 47);
  - Arrays.fill(j, 99);
  - System.arraycopy(i, 0, j, 0, i.length); //static μέθοδος της Java
- Σύγκριση τιμών
  - Arrays.equals(i,j); //ελέγχει ισότητα στα περιεχόμενα των πινάκων
- Ταξινόμηση
  - Arrays.sort(a6);

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***

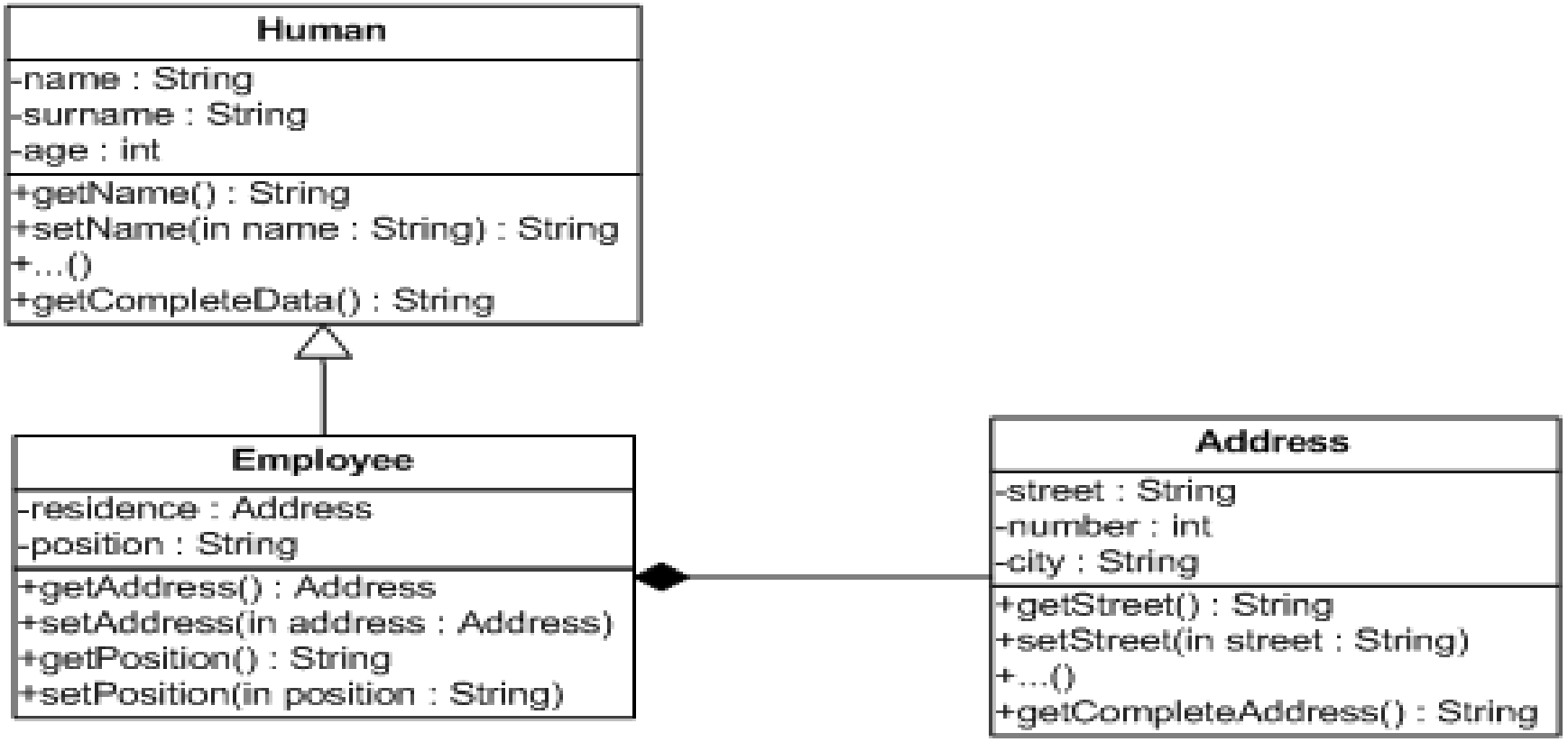
Αφηρημένες τάξεις – abstract classes

- 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.

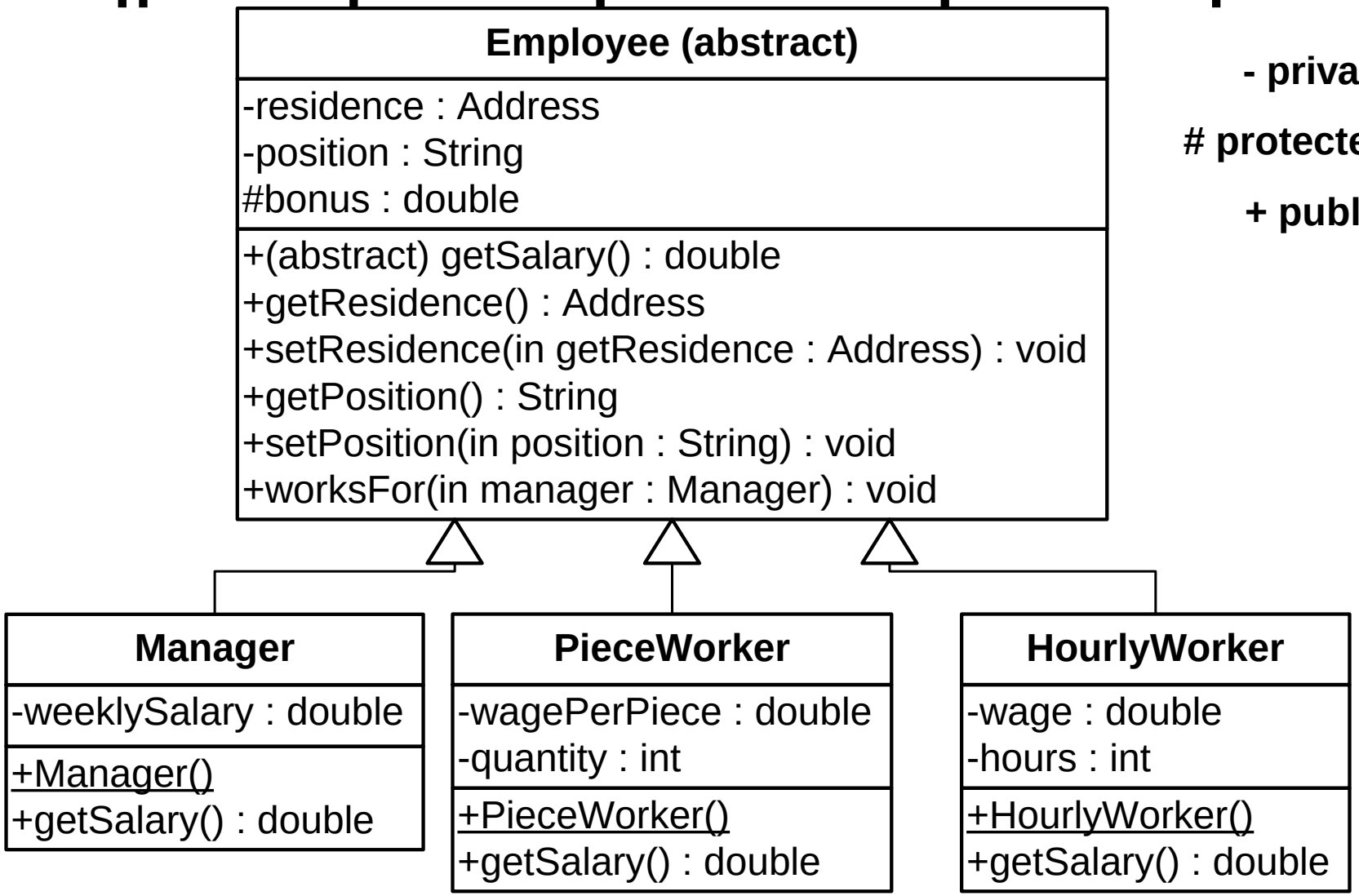
Interface

- 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

Κληρονομικότητα - Παράδειγμα



Κληρονομικότητα και ορατότητα



4ο Μάθημα

- Ρεύματα
  - Προκαθορισμένα ρεύματα εισόδου/εξόδου
- Η τάξη File - Οργάνωση αρχείων
  - Ρεύματα bytes, χαρακτήρων
- Αρχεία κειμένου
  - Ρεύματα εισόδου αρχείων
  - Ρεύματα εξόδου αρχείων
- Διαχείριση λαθών – Εξαιρέσεις
  - Δημιουργία
  - Ανίχνευση
  - Διαχείριση
  - Βασικοί τύποι εξαιρέσεων
  - Δημιουργία τύπων εξαίρεσης

Η κλάση File

- Είναι η βασική κλάση για αρχεία και φακέλους
- Μέθοδοι της `File`
  - `exists`: ελέγχει αν υπάρχει το αρχείο
  - `canRead`: ελέγχει αν μπορούμε να διαβάσουμε από το αρχείο
  - `canWrite`: ελέγχει αν μπορούμε να γράψουμε στο αρχείο
  - `delete`: διαγράφει το αρχείο και επιστρέφει true (αν πετύχει)
  - `length`: επιστρέφει το μέγεθος του αρχείου σε bytes
  - `getName`: επιστρέφει το όνομα του αρχείου (μόνο)
  - `getPath`: επιστρέφει το πλήρες μονοπάτι του αρχείου

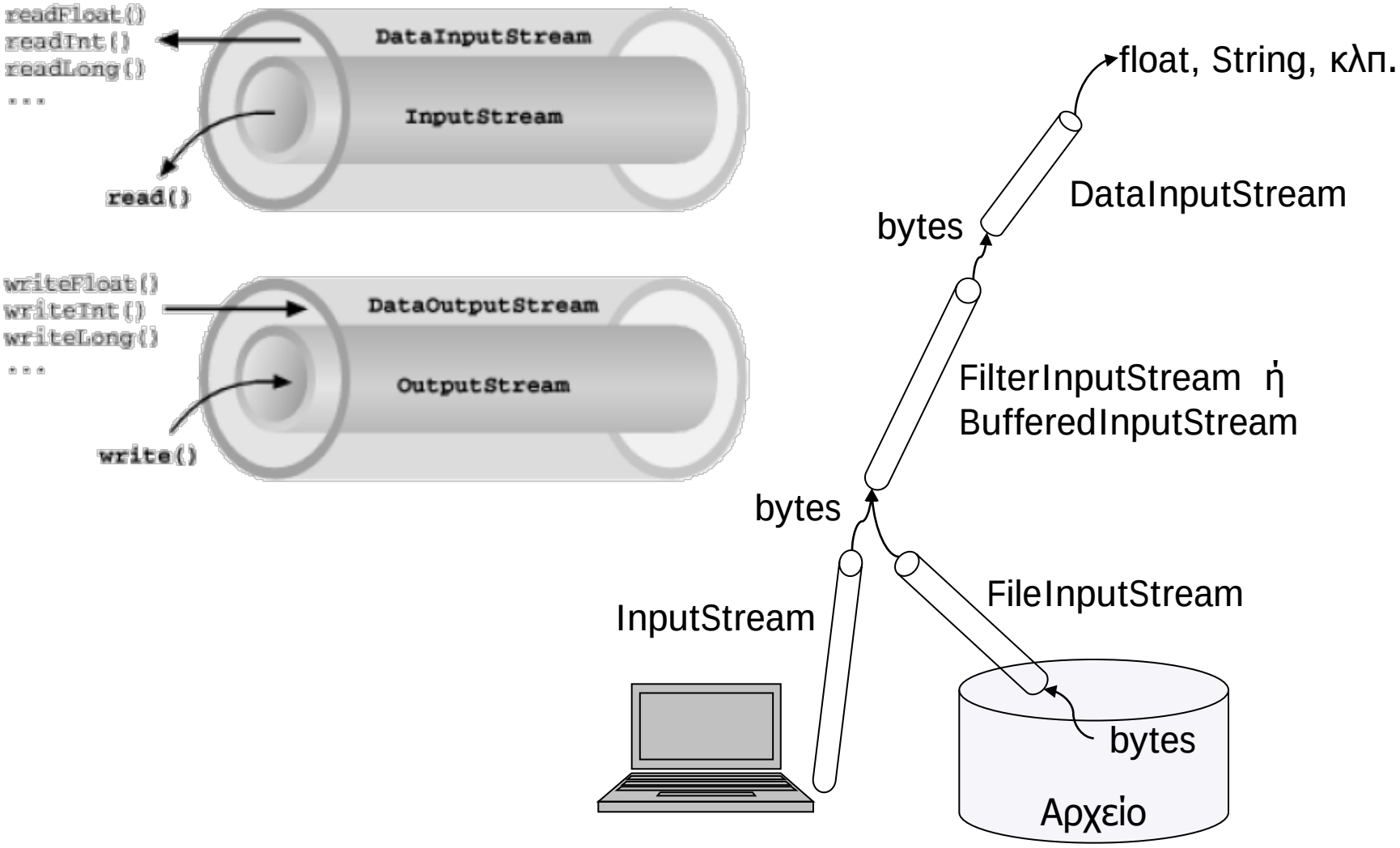
```
File numFile = new File(“numbers.txt”);
if (numFile.exists())
  System.out.println(numfile.length());
```

Σχετικές κλάσεις

- Η FileInputStream και η FileOutputStream έχουν κατασκευαστές που παίρνουν ως όρισμα ένα αντικείμενο File ή το όνομα του αρχείου ως String

```
PrintWriter smileyOutStream = new
  PrintWriter(new
  FileOutputStream("smiley.txt"));
File smileyFile = new File("smiley.txt");
if (smileyFile.canWrite())
  PrintWriter smileyOutStream = new
  PrintWriter(new
  FileOutputStream(smileyFile));
```

Συνδυασμός ρευμάτων bytes



Διάβασμα από αρχεία

- Η τάξη File (στο πακέτο java.io.*) αντιπροσωπεύει:
  - Ένα αρχείο:
    File arxeio = new File("c:\\1.txt");
  - Ένα κατάλογο:
    File katalogos = new File(".");
    String path=katalogos.getAbsolutePath();
    File[] files=katalogos.listFiles();
- Η τάξη FileReader (FileWriter) δημιουργεί ένα ρεύμα εισόδου (εξόδου) από αρχείο:
  - FileReader arxeio = new FileReader("c:\\1.txt"));

Ανάγνωση - Εγγραφή

- Ανάγνωση: Όπως και με το πληκτρολόγιο συνδέουμε το πρόγραμμά μας με το αρχείο με έναν BufferedReader

```
BufferedReader eisodos = new BufferedReader(new
                                FileReader("c:\\1.txt"));
```

- Εγγραφή: Συνδέουμε το πρόγραμμά μας με το αρχείο με ένα BufferedWriter

```
PrintWriter exodos =new PrintWriter(new BufferedWriter(new
                                    FileWriter(“c:\\copy.txt")));
String s;
while((s = eisodos.readLine()) != null )
  exodos.println(lineCount++   + ": " + s);
exodos.close();
```

Αντιμετώπιση λαθών

- Υπάρχουν λάθη χρόνου εκτέλεσης (run time errors) τα οποία δεν ανιχνεύονται στη μεταγλώττιση.
  - Αναζήτηση στοιχείου έξω από τα όρια ενός πίνακα
  - Άνοιγμα αρχείου που δεν υπάρχει
  - Κλήση αναφοράς σε null
- Θα πρέπει να αντιμετωπίζονται όταν το πρόγραμμα εκτελείται
- Ο κώδικας που δημιουργεί κάποιο λάθος μεταδίδει πληροφορία στον κώδικα που καλείται να το αντιμετωπίσει. Αυτό γίνεται με τις ***Εξαιρέσεις – Exceptions***
- Σηματοδοτούν εμφάνιση συνθηκών ιδιαίτερης μεταχείρισης και διακόπτουν τη συνήθη ροή του κώδικα

Αναλυτικά με τις εξαιρέσεις

- Όταν εμφανίζεται μία εξαίρεση σταματά η εκτέλεση της μεθόδου
- Δεν υπάρχει η απαραίτητη πληροφορία στο scope της μεθόδου
- Ανεβαίνουμε και αντιμετωπίζουμε το πρόβλημα σε κάποιο «υψηλότερο» scope. «***Ρίχνουμε***» μια εξαίρεση (throw)
  - Ένα αντικείμενο εξαίρεσης (Exception object) δημιουργείται στον σωρο (heap) με χρήση του τελεστή new όπως κάθε άλλο Java object
  - Διακόπτεται η εκτέλεση της μεθόδου και κρατιέται μια αναφορά στο Exception object σε κάποια περιοχή της μνήμης
  - Αναλαμβάνει ο μηχανισμός διαχείρισης της εξαίρεσης (Exception handling mechanism) που πρέπει να βρει το κατάλληλο μέρος για να συνεχίσει να εκτελεί κώδικα

Αντικείμενα εξαιρέσεων

- Το keyword throw προκαλεί τα ακόλουθα γεγονότα:
  - εκτελεί την έκφραση new και δημιουργεί ένα αντικείμενο που δεν θα υπήρχε κανονικά
  - Το αντικείμενο αυτό επιστρέφεται στην ουσία από την μέθοδο ή το block μέσα στο οποίο δημιουργήθηκε μολονότι η μέθοδος δεν έχει δηλωθεί να επιστρέφει παραμέτρους αυτού του τύπου και τα blocks δεν έχουν φυσικά παραμέτρους που να επιστρέφουν
  - Το πρόγραμμα βγαίνει από τη μέθοδο ή το block.
- Τα αντικείμενα exception που «ρίχνονται» μπορεί να είναι ***διάφορων*** τύπων, ανάλογα το είδος της εξαίρεσης
- Κάθε αντικείμενο κωδικοποιεί στα πεδία του όλη την πληροφορία που αφορά την εξαίρεση, ώστε ο μηχανισμός διαχείρισης εξαιρέσεων στα ψηλότερα επίπεδα να μπορεί να πάρει τις κατάλληλες αποφάσεις

Σύλληψη εξαίρεσης

- Όταν ρίχνουμε μία εξαίρεση πρέπει στα υψηλότερα επίπεδα κάποιο μπλοκ κώδικα να την ανιχνεύσει και να την διαχειριστεί.
- ***Προστατευόμενη περιοχή*** είναι ένα κομμάτι κώδικα που μπορεί να ρίξει εξαιρέσεις και που ακολουθείται από κώδικα που διαχειρίζεται αυτές τις εξαιρέσεις.
- Μια προστατευόμενη περιοχή μπορεί να ανιχνεύσει ένα ή περισσότερους τύπους εξαιρέσεων και
  - να τους διαχειριστεί ή
  - να τους ρίξει ακόμη πιο πάνω (με νέο throw)

Προστατευόμενη περιοχή - *try*

- Ένα τέτοιο μπλοκ ονομάζεται try block αφού εκεί δοκιμάζουμε να κάνουμε κλήσεις σε διάφορες «επικίνδυνες μεθόδους»
- Το try block είναι ένα κανονικό scoper που περικλείεται από την κωδική λέξη try

```
try {
   // κώδικας που παράγει εξαιρέσεις
}
```

- Προστασία από εξαιρέσεις: Βάζουμε όλον τον «επικίνδυνο» κώδικα σε ένα try block και πιάνουμε όλες τις εξαιρέσεις στο ίδιο μέρος.

Διαχειριστές εξαιρέσεων - ***catch***

- Κάθε εξαίρεση που ανιχνεύεται μέσα στο try καταλήγει στον αντίστοιχο κώδικα (***catch*** block) που θα την εξυπηρετήσει. Τα catch ελέγχονται διαδοχικά.
- Παράδειγμα:

```
try {
   // κώδικας που παράγει εξαιρέσεις
} catch (type1 id1) {
   // χειρισμός εξαιρέσεων τύπου 1
} catch (type2 id2) {
   // χειρισμός εξαιρέσεων τύπου 2
} catch (type3 id3) {
   // χειρισμός εξαιρέσεων τύπου 3
}
```

- Αν πολλές μέθοδοι στο try ρίχνουν τον ίδιο τύπο εξαίρεσης, τότε χρειαζόμαστε μόνο έναν exception handler για αυτόν τον τύπο.

Παράδειγμα – έλεγχος εισόδου (2)

- Μπορούμε να χρησιμοποιήσουμε δική μας εξαίρεση;
- Στο αρχείο MySpecialException.java

```
public class MySpecialException extends Exception{

}
```

- Οπότε

```
public double readDouble() throws MySpecialException{
      Scanner in =new Scanner(System.in);
       if(in.hasNextDouble()){
           return in.nextDouble();
       }
       else throw new MySpecialException();
}
```

Παράδειγμα

- Μη διαχείριση σφάλματος εισόδου

```
public double sumNumbers() throws MySpecialException{
      double sum=0;
      sum+=readDouble();
      return sum;
}
 public static void main(String[] args) throws MySpecialException{
        Main m=new Main();
        double z=m.sumNumbers();
        System.out.println("Sum is:"+z);
    }
```

- Έτσι αποφεύγουμε να χειριστούμε την εξαίρεση

Παράδειγμα

- Διαχείριση σφάλματος εισόδου

```
public double sumNumbers(){
      double sum=0;
       try{
                sum+=readDouble();
       }
       catch(MySpecialException ex){              //θα συμβεί αν ο χρήστης δεν δώσει αριθμό
            System.err.println(ex.toString());    //τυπώνει το όνομα της κλάσης εξαίρεσης
       }
       catch(Exception ex){                       //θα συμβεί σε οποιαδήποτε άλλη περίπτωση λάθους
           System.err.println(ex.getMessage());   //τυπώνει το μήνυμα της εξαίρεσης
       }
      return sum;
}
```

- Η MySpecialException είναι ειδικότερη από την Exception

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");
}
```

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.

Υπέρβαση μεθόδων

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

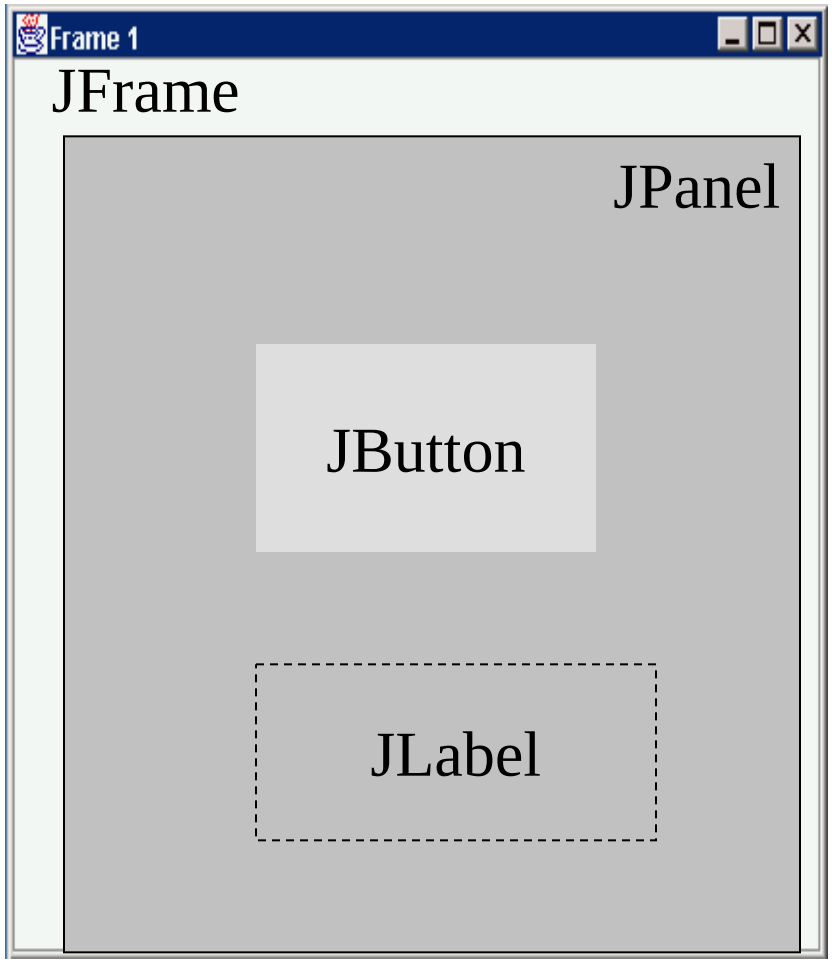
```
class Employee{
...
String getCompleteData(){
    String data=new String();
    data=super.getCompleteData() + " "+
         residence.getCompleteAddress()+" position:"+
         position;
    return data;
}
}
```

7ο Μάθημα

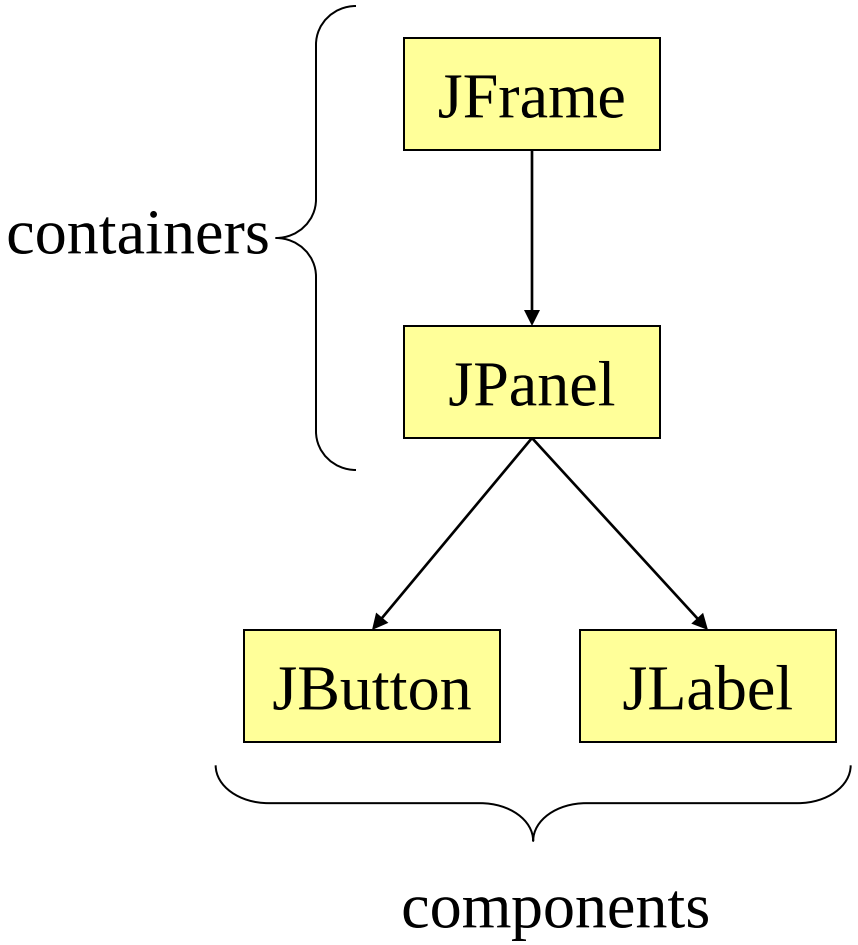
- Γραφικά περιβάλλοντα (GUI)
- Abstract Windowing Toolkit (AWT)
  - Containers
  - Components
  - Layout managers
  - (Listeners)
- Swing
  - Αρχιτεκτονική Model-View-Controller
  - Διάφορα στοιχεία του Swing
  - Παράθυρα και μενού
  - Περιγράμματα
  - Τοποθέτηση στοιχείων στο παράθυρο
- Διαχειριστές τοποθέτησης
  - Βασικά interfaces
  - Τάξεις

Τα μέρη ενός γραφικού περιβάλλοντος

GUI

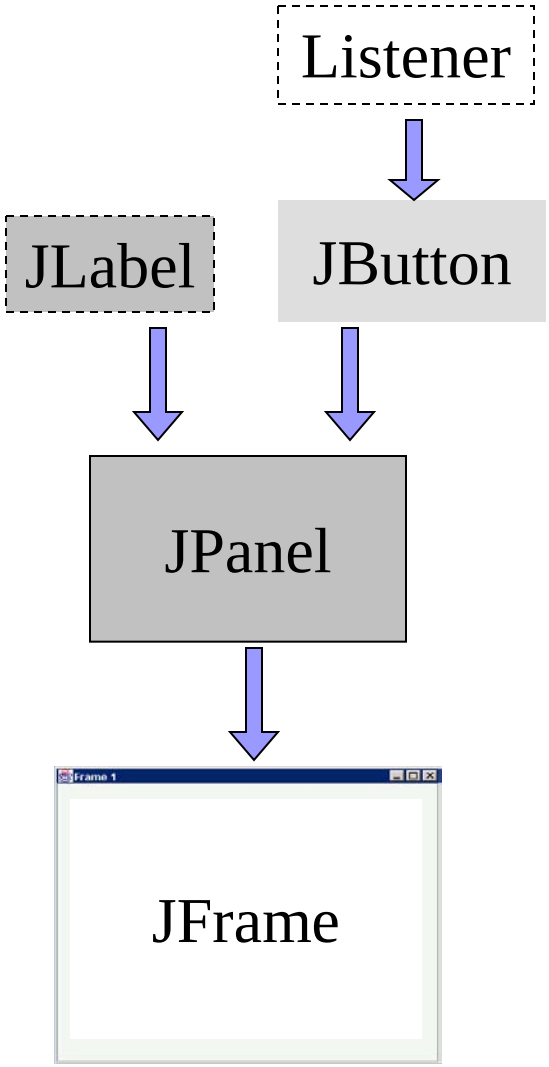


Σύνθεση τάξεων



Με ποια σειρά φτιάχνουμε το GUI

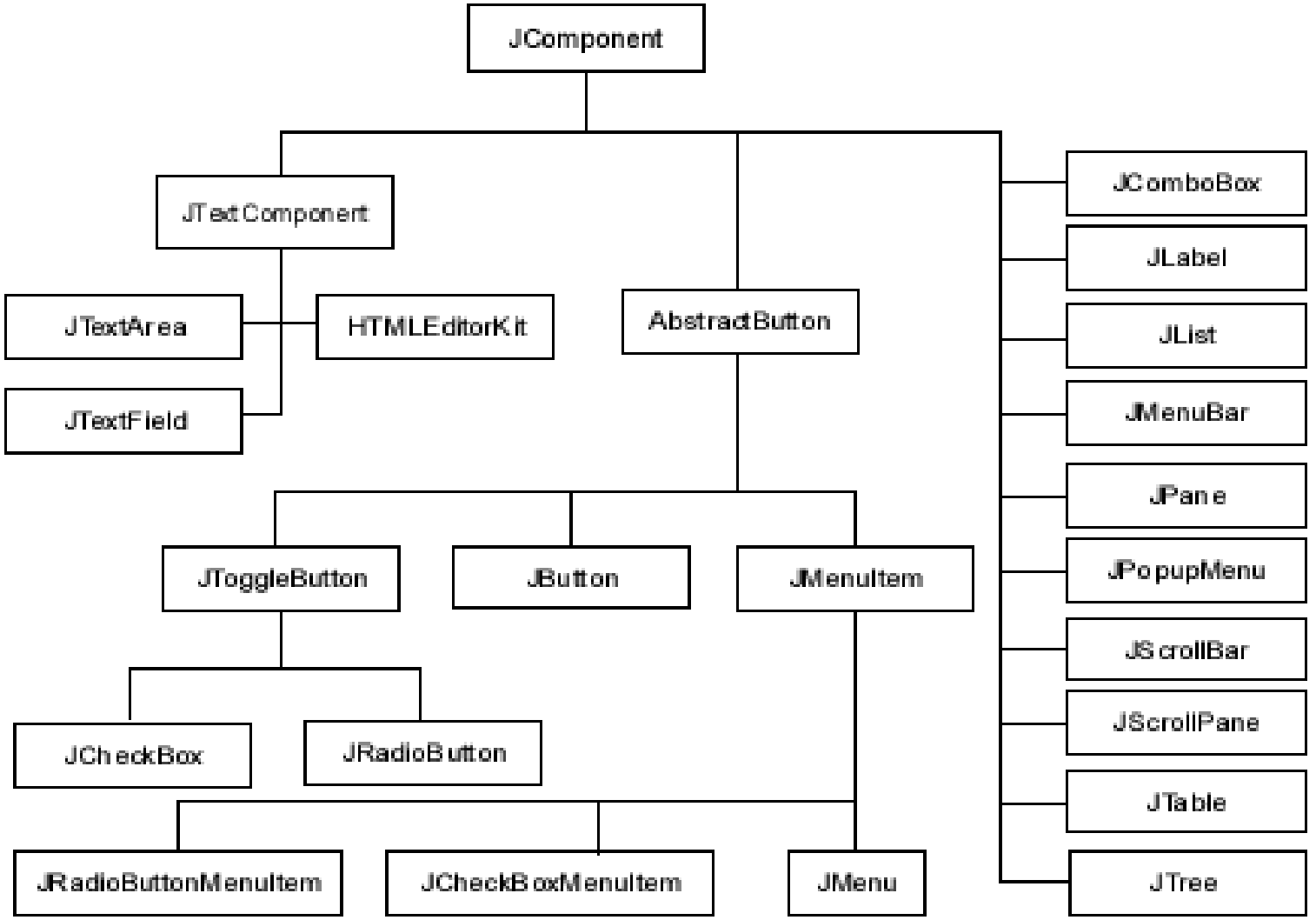
- Δημιουργούμε
  - Το JFrame
  - Το JPanel
  - Τα Components (JButton, JLabel)
  - Το Listener για το JButton
- Προσθέτουμε (μέθοδος add)
  - Τον Listener στο JButton
  - Τα components στο Jpanel
  - Το JPanel στο JFrame
- Εμφανίζουμε
  - Το JFrame (μέθοδος show)



8ο Μάθημα

- Στοιχεία
  - Περιοχές κειμένου
  - Κουμπιά
  - Λίστες επιλογών
- Παράθυρα διαλόγου
  - Παράθυρα επιβεβαίωσης
  - Παράθυρο επιλογής αρχείου
  - Παράθυρο επιλογής χρώματος

Στοιχεία του Swing



Λίστες επιλογών

- JComboBox: Περιέχει ένα πίνακα από επιλογές, εμφανίζει στο χρήστη μόνο μία και επιτρέπει μονές ή πολλαπλές επιλογές.

```
String [] items = { "uno", "due", "tre", "quattro", "cinque", "sei", "sette",
    "otto", "nove", "deici", "undici" };
JComboBox comboBox = new JComboBox(items);
comboBox.addItem("dodici");
comboBox.getSelectedItem() //επιστρέφει Object
comboBox.getSelectedObjects() //επιστρέφει Object[]
```

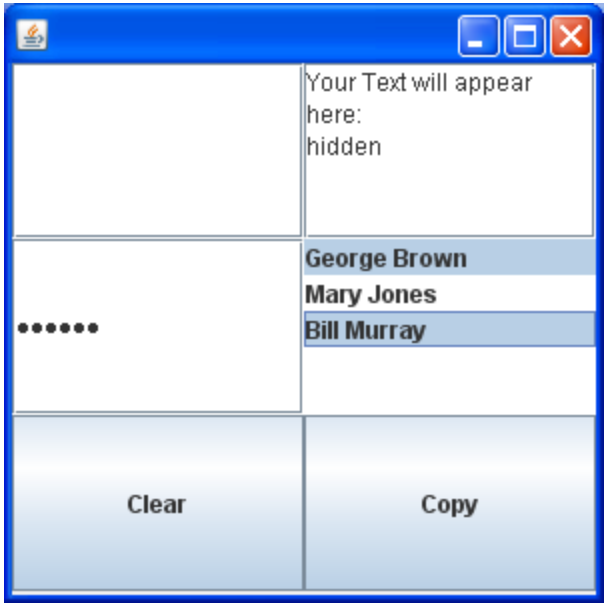
- JList: Περιέχει ένα πίνακα από επιλογές, εμφανίζει στο χρήστη ορισμένες από αυτές (ανάλογα με το ύψος της) και επιτρέπει μονές ή πολλαπλές επιλογές.

```
JList list = new JList(comboBox.getModel( ));
list.getSelectedValues() //επιστρέφει Object[]
```

- Μοιράζονται το ίδιο μοντέλο δεδομένων

Μοντέλα Δεδομένων για Λίστες

```
dlm = new DefaultListModel();
dlm.addElement(new Human("George", "Brown", 22, "6th Avenue"));
dlm.addElement(new Human("Mary", "Jones", 18, "5th Avenue"));
dlm.addElement(new Human("Bill", "Murray", 19, "Madison Avenue"));
students = new JList(dlm);
this.getContentPane().add(students);
this.getContentPane().add(clearButton);
this.getContentPane().add(copyButton);
```



- Φαίνεται ότι επιστρέφει η toString()
- Επιστρέφει όλο το Object

```
Object[] selected=students.getSelectedValues();
for (int i=0;i<selected.length;i++)
    Human h=(Human)selected[i];
```

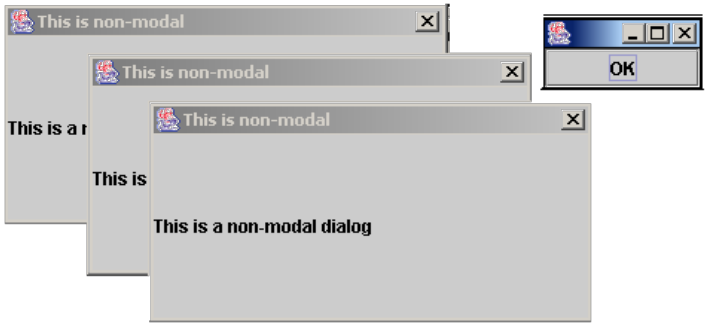
Το παράθυρο JDialog

- Συμπεριφέρεται όπως και το JFrame

```
JDialog dialog = new JDialog(myFrame, "Dialog Frame");
JLabel label = new JLabel("This is a message“);
dialog.getContentPane().add(label);
dialog.setVisible(true);
```

- Μπορεί να μπλοκάρει τη συνέχιση του προγράμματος (modal) ή όχι (non-modal)

```
JDialog dialog1 = new JDialog(myFrame, "This is modal", true);
JDialog dialog2 = new JDialog(myFrame, "This is non modal", false);
```



9ο Μάθημα

- Συλλογές και ενέργειες
- ArrayList
  - Αναζήτηση συγκεκριμένου στοιχείου
  - Αναζήτηση ελαχίστου/μεγίστου
  - Συνάθροιση
- Πίνακας
  - Εισαγωγή
  - Εισαγωγή με έλεγχο διπλοτύπων

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

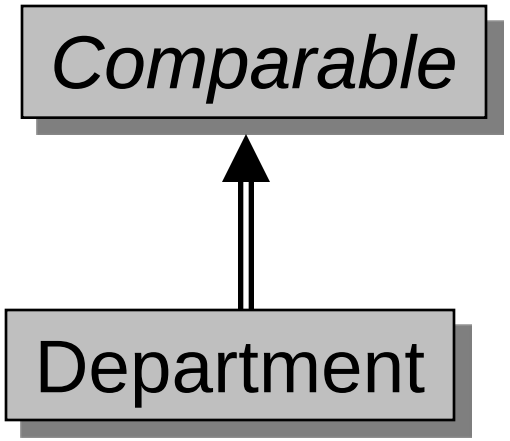
- Με ποιο τρόπο μπορώ να ταξινομήσω τα στοιχεία ενός πίνακα ή μιας λίστας; BubbleSort:

```
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

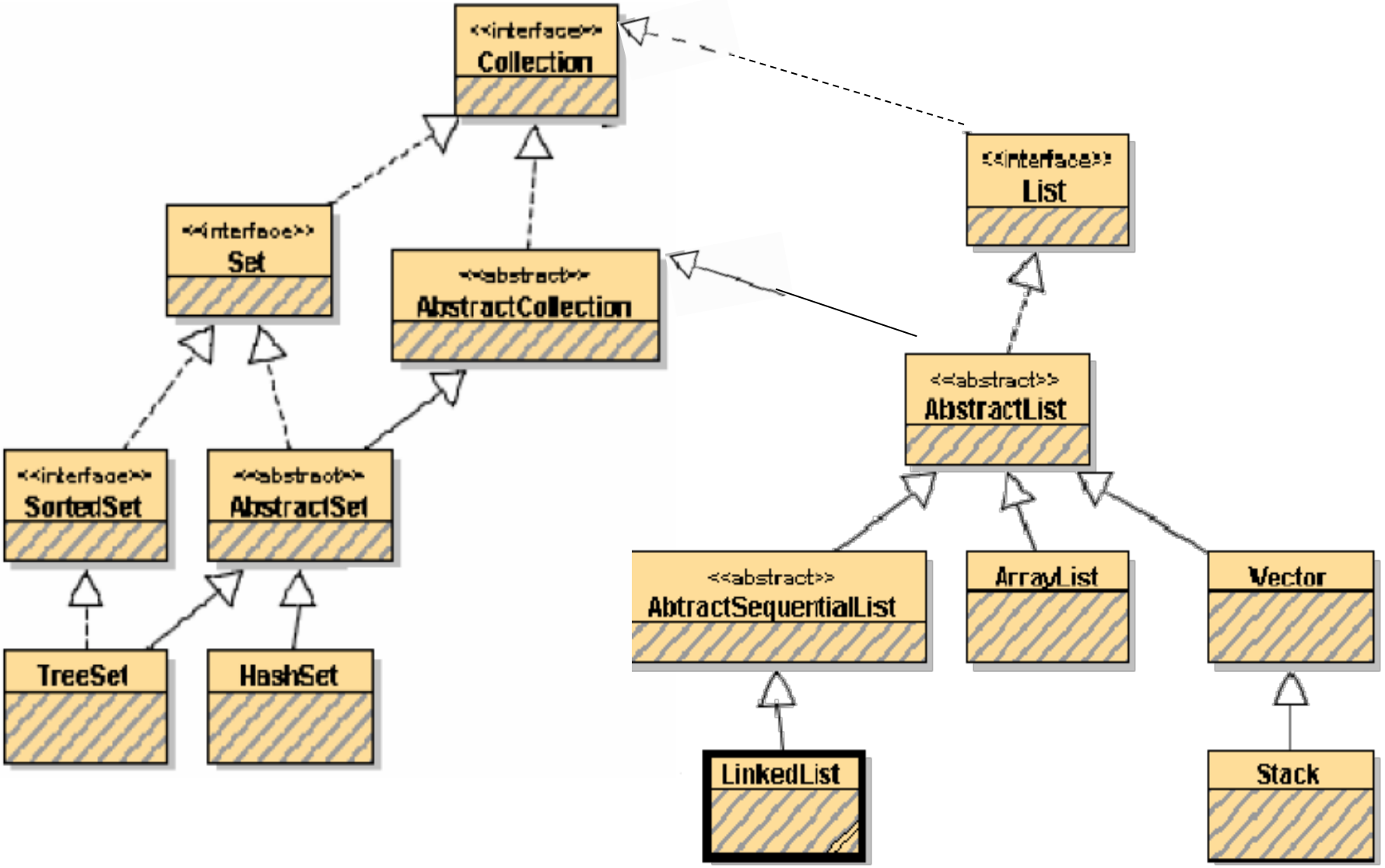
Χρησιμοποιεί την QuickSort

Διεπαφές – υλοποιήσεις και παρελθόν

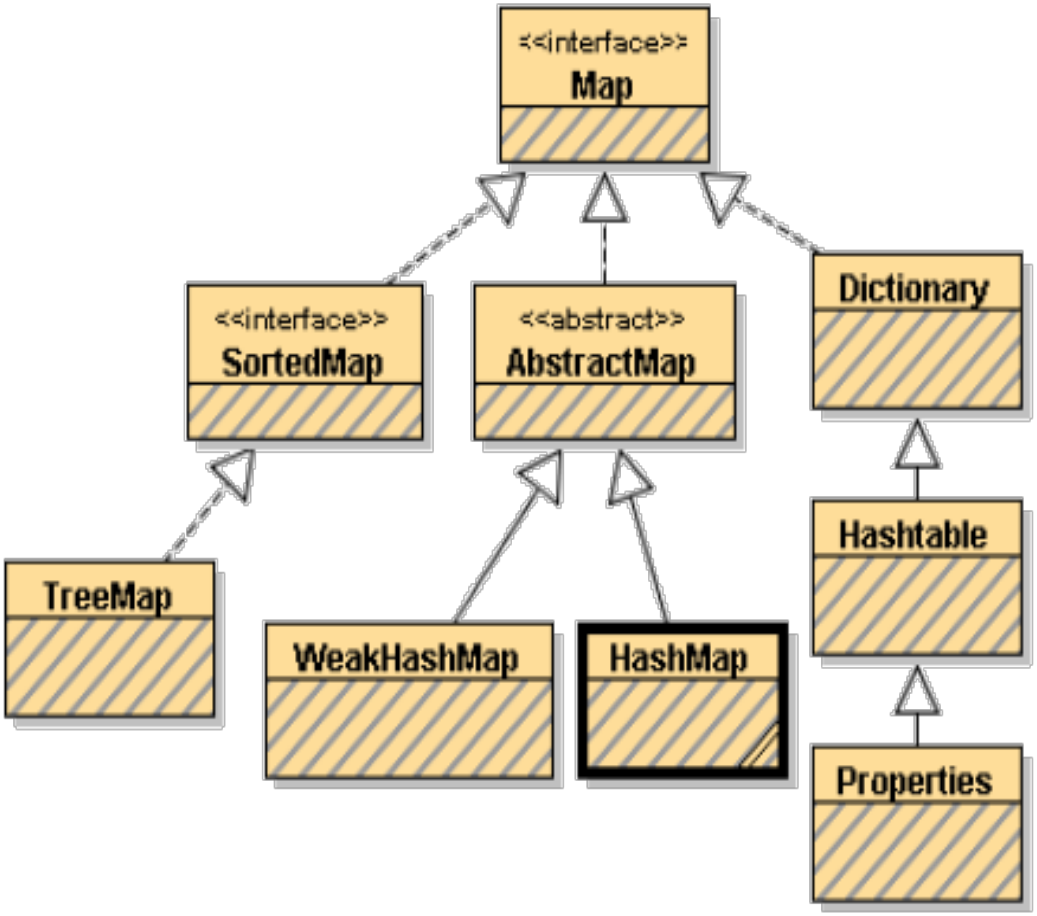
Διεπαφή	Υλοποιήσεις				Παλιότερες υλοποιήσεις
Set	HashSet		TreeSet		
List		ArrayList		LinkedList	Vector Stack
Map	HashMap		TreeMap		HashTable Properties

Μια τάξη υλοποίησης μπορεί να μην υλοποιεί μια συγκεκριμένη μέθοδο του interface (κάποιες μέθοδοι είναι προεραιτικές)
Στην περίπτωση αυτή παράγει **UnsupportedOperationException**

Υλοποιήσεις



Υλοποιήσεις



- WeakHashMap είναι μια υλοποίηση του Map που καλεί τον garbage collector όταν ένα κλειδί δε χρησιμοπείται πλέον

11ο Μάθημα

- Πολυνηματικές εφαρμογές
  - Ορισμοί
  - Έλεγχος νήματος
  - Συγχρονισμός νημάτων

Threads - Νήματα

- Τα threads επιτρέπουν εργασίες να τρέχουν παράλληλα με την κύρια εφαρμογή
- Είναι διεργασίες που τρέχουν στον ίδιο χώρο διευθύνσεων: μοιράζονται τις μεταβλητές στιγμιοτύπων, αλλά διατηρούν και τοπικές μεταβλητές
- Χρησιμοποιούνται από το JVM σε συγκεκριμένες περιπτώσεις
  - Όταν φορτώνουμε μια εικόνα ή ένα ήχο μέσα από ένα applet.
  - Όταν καλούμε την update του αντικειμένου graphics
- Μπορούμε να δηλώσουμε και να χρησιμοποιήσουμε δικά μας νήματα.

Τρόποι δήλωσης (1/2)

- Επέκταση της Thread

```
public class HelloThread extends Thread {
    public void run() {
        System.out.println("Hello from a thread!");
    }
    public static void main(String args[]) {
        (new HelloThread()).start();
    }
}
```

Τρόποι δήλωσης (2/2)

- Υλοποίηση της Runnable

```
public class HelloRunnable implements
Runnable {
    public void run() {
      System.out.println("Hello from a thread!");
    }
    public static void main(String args[]) {
      (new Thread(new HelloRunnable())).start();
    }
 }
```

Μέθοδοι

- run(): το βασικό σύνολο εντολών του νήματος. Ενεργοποιείται με το start()
- static void sleep(milliseconds): σταματά προσωρινά την εκτέλεση του νήματος
- void interrupt(): διακόπτει το νήμα
- boolean interrupted(): ελέγχει αν το νήμα εξακολουθεί να είναι σταματημένο ή ξαναξεκίνησε
- final void otherThread.join(): διακόπτει το τρέχον νήμα μέχρι να τελειώσει το otherThread

12ο μάθημα

- Επικοινωνία με δικτυακές πηγές
- Ανάγνωση από πηγή

```
BufferedReader in = new BufferedReader( new
   InputStreamReader( target.openStream()));
String inputLine;
while ((inputLine = in.readLine()) != null)
        System.out.println(inputLine);
in.close();
```

Σύνδεση με πηγή

```
try {
    URL hua = new URL("http://www.hua.gr/");
    URLConnection huaConnection = hua.openConnection();
    huaConnection.connect();
    BufferedReader in = new BufferedReader( new InputStreamReader(
    huaConnection.getInputStream()));
    …
}
catch (MalformedURLException e) {
// new URL() failed . . .
}
catch (IOException e) {
// openConnection() failed . . .
}
```

Γράφοντας σε μια διεύθυνση

- Δημιουργία του URL

```
URL url = new URL("http://www.hua.gr");
```

- Ανάκτηση του URLConnection

```
URLConnection connection = url.openConnection();
```

- Ενεργοποίηση εξόδου

```
connection.setDoOutput(true);
```

- Δημιουργία ρεύματος εξόδου

```
OutputStreamWriter out = new OutputStreamWriter(
    connection.getOutputStream());
```

- Πέρασμα παραμέτρων

```
out.write("id=25");
```

- Κλείσιμο του ρεύματος

```
out.close();
```