Οντοκεντρικός Προγραμματισμός (lect 1)

ΦΡΟΝΤΙΣΤΗΡΙΟ JAVA

Μαρία Ρήγκου
rigou@ceid.upatras.gr

Διαδικαστικά Μαθήματος

Θεωρία – Φροντιστήριο - Εργαστήριο

ΕΒΔΟΜΑΔΙΑΙΟ ΠΡΟΓΡΑΜΜΑ

- Θεωρία
  - Δευτέρα 16.00-19.00 (ΒΑ)
- Φροντιστήριο
  - Πέμπτη 11.00-13.00 (ΑΠ7)
- Εργαστήριο (έναρξη: εβδομάδα 29/2/15)
  - Δευτέρα: 09.00-11.00, 11.00-13.00, 13.00-15.00
  - Τετάρτη: 9.00-11.00, 11.00-13.00

Εγγραφείτε σε μια από τις ομάδες στο eclass μέχρι και Σάββατο 27/2

ΕΡΓΑΣΤΗΡΙΟ

- 3 εργαστήρια (2 Java, 1 C++), 3 εβδομάδες το καθένα
- Κάθε εργαστήριο περιλαμβάνει:
  - Ένα σετ εργαστηριακών ασκήσεων εκτέλεσης (από φυλλάδιο)
  - Ένα σετ ασκήσεων σχεδίασης και εκτέλεσης
- Οι δύο πρώτες εβδομάδες κάθε εργαστηρίου είναι εβδομάδες προετοιμασίας/εξάσκησης
- Την τρίτη εβδομάδα παραδίδετε τεχνική αναφορά και για τα δύο σετ και εξετάζεστε προφορικά (πρόοδος)

ΑΞΙΟΛΟΓΗΣΗ

- Πρόοδοι στην ύλη του εργαστηρίου
  - Υποχρεωτικές (χάνεται το αντίστοιχο ποσοστό σε περίπτωση μη προσέλευσης): τρεις πρόοδοι + τελική εξέταση εργαστηρίου
  - Βαρύτητα: 5%+5%+5%+15%=30%
- Εργασία (Project)
  - Ένα σε Java (10%)
  - Αντίστοιχο σε C++ (10%)
- Γραπτή εξέταση
  - Βαρύτητα: 50%

ΑΞΙΟΛΟΓΗΣΗ

- Προϋπόθεση συμμετοχής στη γραπτή εξέταση
  - Να έχετε βαθμολογηθεί στην τελική εξέταση εργαστηρίου και σε 1 τουλάχιστον από τις 3 προόδους εργαστηρίου
  - Να έχετε βαθμολογηθεί με τουλάχιστον 4.0 σε ένα από τα δύο project
- Προϋπόθεση προσμέτρησης εργαστηρίων και project
  - Να έχετε βαθμολογηθεί στην γραπτή εξέταση με τουλάχιστον 4.5

ΠΛΗΡΟΦΟΡΙΕΣ

Ιστοσελίδα εργαστηρίου

https://eclass.upatras.gr/courses/CEID1105/

Εδώ βρίσκεται αναρτημένο και το φυλλάδιο εργαστηρίου

Δείτε προσεκτικά τη σελίδα ΠΛΗΡΟΦΟΡΙΕΣ

Contact info:

[Φροντιστήριο/Εργαστήριο]

Μαρία Ρήγκου

rigou@ceid.upatras.gr

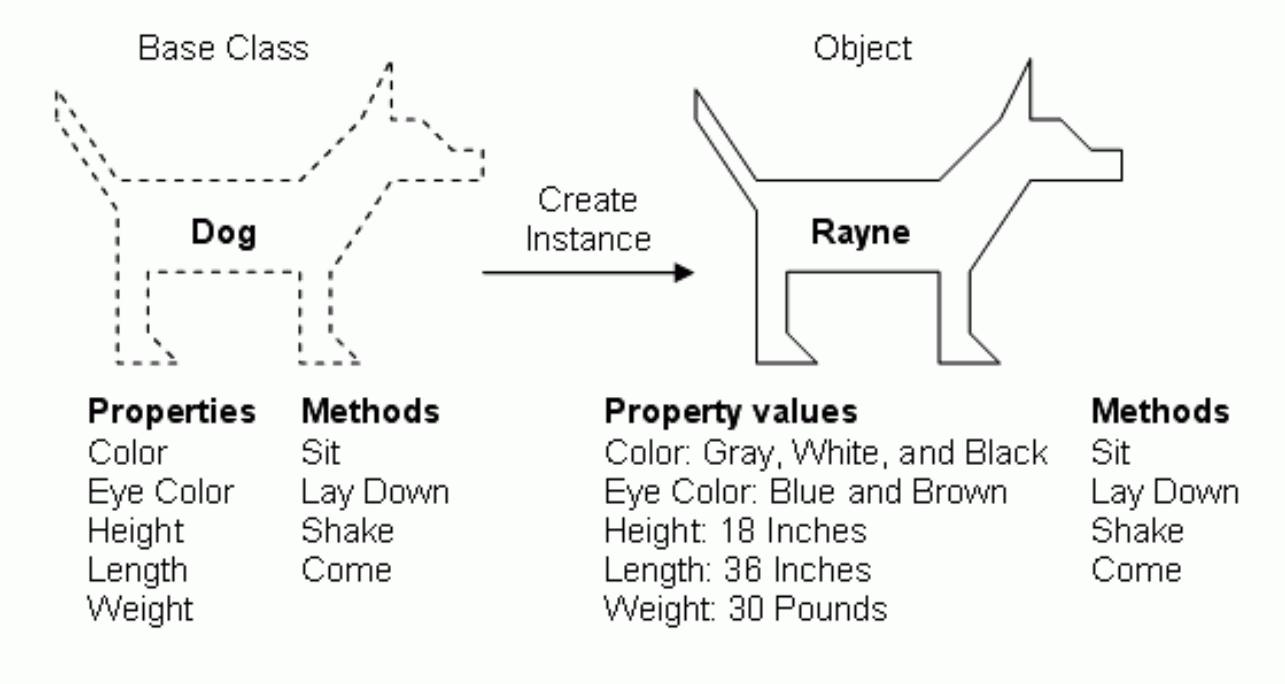
[Εργαστήριο]

Κόβας Κων/νος

kobas@ceid.upatras.gr

BlueJ

Αντικειμενοστρεφής τρόπος σκέψης



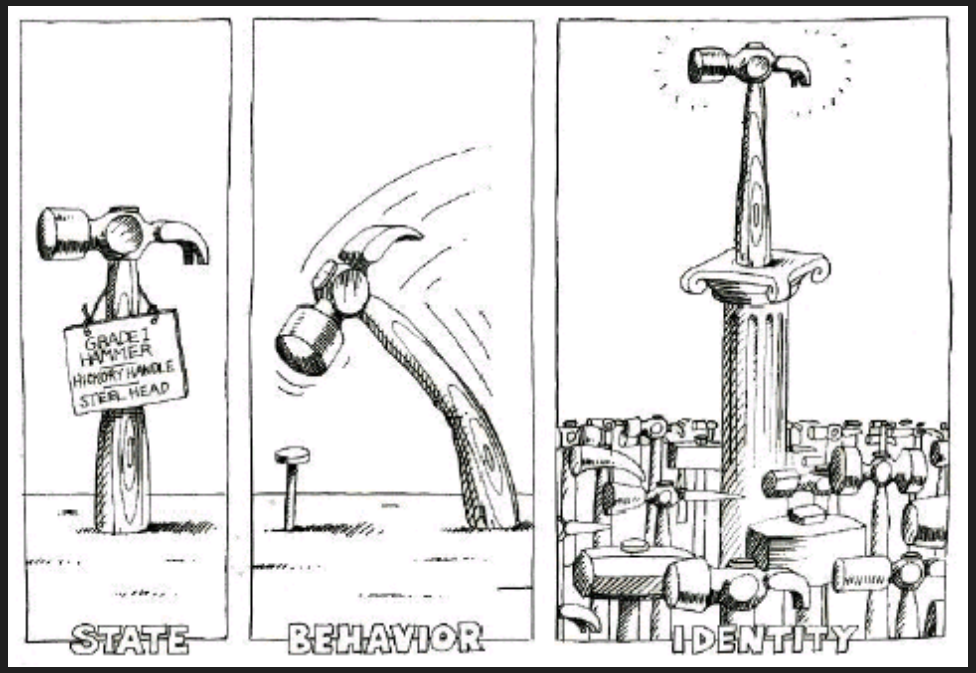
Βασικές αρχές αντικειμενοστρέφειας (αρχές του Allan Kay)

- Τα πάντα είναι αντικείμενα
- Κάθε αντικείμενο έχει δικιά του μνήμη
- Κάθε αντικείμενο έχει ένα συγκεκριμένο τύπο
  - Τύπος = Κλάση
- Τα αντικείμενα επικοινωνούν μέσω μηνυμάτων
- Αντικείμενα του ίδιου τύπου μπορούν να δεχτούν τα ίδια μηνύματα
  - Δηλαδή μπορούν να εκτελούν τις ίδιες λειτουργίες
- Ένα πρόγραμμα είναι μια συλλογή από αντικείμενα όπου το ένα λέει στο άλλο τι να κάνει

Αντικείμενο

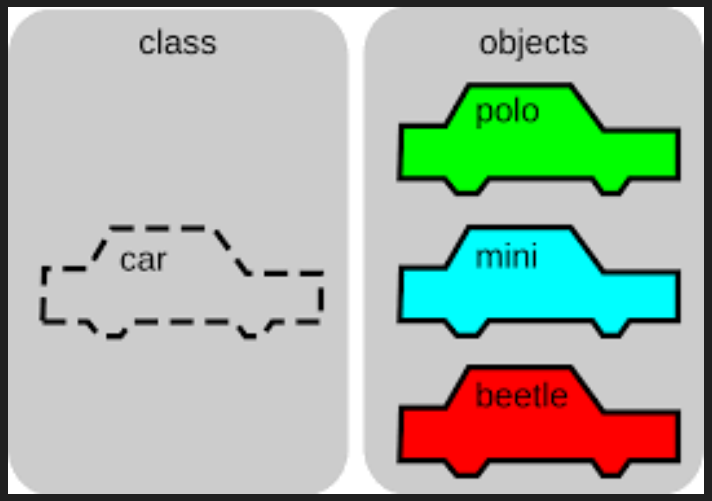
- Ένα αντικείμενο στον κώδικα αναπαριστά μια οντότητα (έννοια) και έχει:

- Μια κατάσταση, η οποία ορίζεται από ορισμένα χαρακτηριστικά
- Μια συμπεριφορά, η οποία ορίζεται από ορισμένες ενέργειες που μπορεί να εκτελέσει το αντικείμενο
- Μια ταυτότητα που το ξεχωρίζει από τα υπόλοιπα.



Παραδείγματα: ένας άνθρωπος, ένα πράγμα, ένα μέρος, μια υπηρεσία

Κλάσεις



- Κλάση: Μια αφηρημένη περιγραφή αντικειμένων με κοινά χαρακτηριστικά και κοινή συμπεριφορά
  - Ένα καλούπι που παράγει αντικείμενα
  - Ένα αντικείμενο είναι ένα στιγμιότυπο μίας κλάσης.
- Π.χ., η κλάση φοιτητής έχει τα γενικά χαρακτηριστικά (όνομα, βαθμοί) και τη συμπεριφορά print
  - Ο φοιτητής Χ είναι ένα αντικείμενο της κλάσης φοιτητής
- Η κλάση Car έχει τα χαρακτηριστικά (brand, color) και τη συμπεριφορά (drive, stop)
  - Το αυτοκίνητο AXX2013 είναι ένα αντικείμενο της κλάσης Car με κατάσταση τα χαρακτηριστικά (BMW, red)

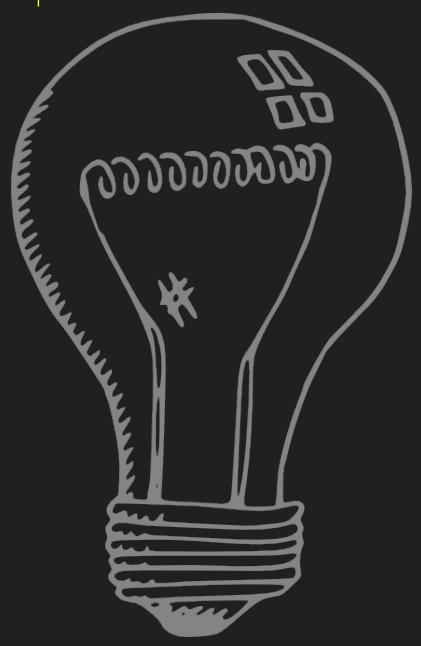
Πρακτικά στον κώδικα

- Μία κλάση Κ ορίζεται από
  - Κάποιες μεταβλητές τις οποίες ονομάζουμε πεδία
  - Κάποιες συναρτήσεις που τις ονομάζουμε μεθόδους.
    - Οι μέθοδοι «βλέπουν» τα πεδία της κλάσης

μέλη της κλάσης

- Ένα αντικείμενο ορίζεται ως μια μεταβλητή τύπου Κ
  - Στην Java (όπως και στις περισσότερες γλώσσες) όλες οι μεταβλητές έχουν ένα τύπο ("strongly typed").
  - Το αντικείμενο που δημιουργείται παίρνει κάποιες τιμές στα πεδία της κλάσης και καταλαμβάνει κάποιο χώρο στη μνήμη.
    - Έτσι μετατρέπεται σε ένα φυσικό αντικείμενο με μοναδική ταυτότητα.

Δημιουργώντας φως



Θέλουμε να κάνουμε ένα πρόγραμμα που να διαχειρίζεται τα φώτα σε διάφορα δωμάτια και θα υλοποιεί και ένα dimmer

Light	Όνομα κλάσης
intensity	Πεδία κλάσης
on() off() dim() brighten()	Μέθοδοι κλάσης

Αντικείμενα:

Light bedroomLight
Light kitchenLight

Τα αντικείμενα δημιουργούνται σε άλλο σημείο του κώδικα το οποίο καλεί και τις μεθόδους

Η κλήση μιας μεθόδου για ένα αντικείμενο μερικές φορές λέγεται και πέρασμα μηνύματος

Πλεονεκτήματα Object Orientation

- Τα αντικείμενα και οι κλάσεις μοντελοποιούν με φυσικό τρόπο τα αντικείμενα του κόσμου
- Τα πλεονεκτήματα είναι ότι αυτό κάνει τον κώδικα
  - πιο ευανάγνωστο
  - πιο τμηματοποιημένο
  - και πιο εύκολο να συντηρηθεί.

Παράδειγμα

- Θέλουμε να προσομοιώσουμε ένα αυτοκίνητο το οποίο κινείται πάνω σε μία ευθεία. Αρχικά ξεκινάει από τη θέση μηδέν. Σε κάθε χρονική στιγμή κινείται τυχαία κατά μία θέση είτε αριστερά είτε δεξιά. Σε κάθε βήμα τυπώνεται μια κουκίδα που δείχνει τη θέση του.

- Πώς θα λύσουμε αυτό το πρόβλημα?
  - Τι κλάσεις και τι αντικείμενα θα ορίσουμε?
  - Τι πεδία και τι μεθόδους θα έχουν?

Αντικειμενοστρεφής Σχεδίαση

- Οι οντότητες/έννοιες στον ορισμό του προβλήματος γίνονται κλάσεις και ορίζονται τα αντικείμενα που αναφέρονται στο πρόβλημα.
- Τα ρήματα γίνονται μέθοδοι
- Τα χαρακτηριστικά των αντικειμένων γίνονται πεδία
  - Τα πεδία μπορεί να είναι κι αυτά αντικείμενα.

- Δεν υπάρχει ένας μοναδικός τρόπος να μοντελοποιήσετε ένα πρόβλημα. Συνήθως όμως υπάρχει ένας που είναι καλύτερος από τους άλλους.

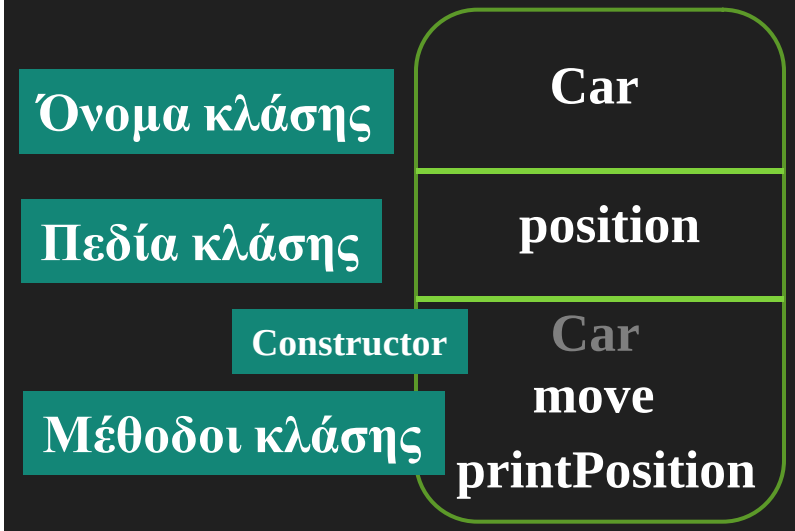
Παράδειγμα

- Θέλουμε να προσομοιώσουμε ένα αυτοκίνητο το οποίο κινείται πάνω σε μία ευθεία. Αρχικά ξεκινάει από τη θέση μηδέν. Σε κάθε χρονική στιγμή κινείται τυχαία κατά μία θέση είτε αριστερά είτε δεξιά. Σε κάθε βήμα τυπώνεται μια κουκίδα που δείχνει τη θέση του.

- Πώς θα λύσουμε αυτό το πρόβλημα?
  - Τι κλάσεις και τι αντικείμενα θα ορίσουμε?
  - Τι πεδία και τι μεθόδους θα έχουν?

Υλοποίηση

Αν έχω δύο αυτοκίνητα?

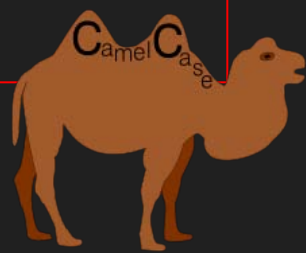


Πρόγραμμα

- Δημιούργησε το αντικείμενο myCar τύπου Car
  - Car myCar = new Car()
- myCar.printPosition()
- For i = 1…10
  - myCar.move()
  - myCar.printPosition()

Programming Style

- Τα ονόματα των κλάσεων ξεκινάνε με κεφαλαίο, τα ονόματα των πεδίων, μεθόδων και αντικειμένων με μικρό.
- Χρησιμοποιούμε ολόκληρες λέξεις (και συνδυασμούς τους) για τα ονόματα
  - Δεν πειράζει αν βγαίνουν μεγάλα ονόματα
- Χρησιμοποιούμε το CamelCase Style
  - Όταν για ένα όνομα έχουμε πάνω από μία λέξη, τις συνενώνουμε και στο σημείο συνένωσης κάνουμε το πρώτο γράμμα της λέξης κεφαλαίο
  - printPosition όχι print_position
- Χρησιμοποιούμε κεφαλαία και '_' για τις σταθερές.



Απόκρυψη - Ενθυλάκωση

- Στο πρόγραμμα που κάναμε πριν δεν είχαμε πρόσβαση στην θέση του αυτοκινήτου
  - Μόνο οι μέθοδοι της κλάσης μπορούν να την αλλάξουν.
  - Γιατί?
    - Αν μπορούσε να αλλάζει σε πολλά σημεία στον κώδικα τότε κάποιες άλλες μέθοδοι θα μπορούσαν να το αλλάξουν και να δημιουργηθεί μπέρδεμα
    - Τώρα αλλάζει μόνο όταν είναι λογικό να αλλάξει – όταν κινηθεί το όχημα.
- Κάποιος που χρησιμοποιεί τις μεθόδους της κλάσης δεν ξέρει πως υλοποιούνται, γνωρίζει μόνο τι κάνουν και πως μπορεί να τις καλέσει
- Αυτή η αρχή λέγεται Ενθυλάκωση – Απόκρυψη Πληροφορίας

Παράδειγμα

- Τι γίνεται αν στο αρχικό μας παράδειγμα αν εκτός από αυτοκίνητο είχαμε και μία μηχανή? Η μηχανή κινείται με τον ίδιο τρόπο αλλά όταν τυπώνεται η θέση της αντί για κουκίδα (.) τυπώνεται ένα αστέρι (*).

- Τι κλάσεις πρέπει να ορίσουμε?

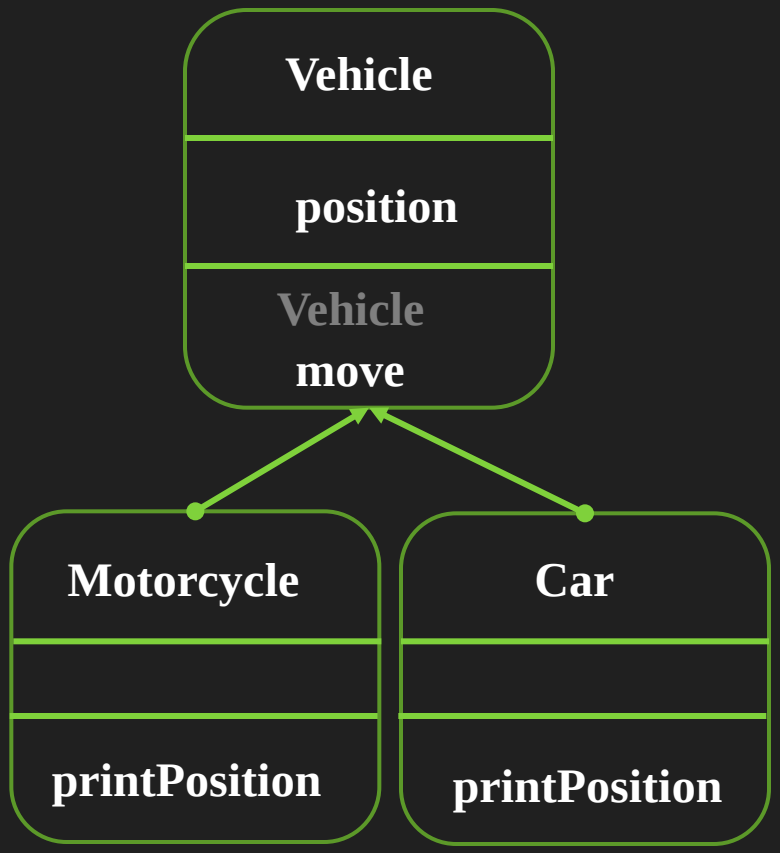
Μία λύση

- Μπορούμε να ορίσουμε ξανά από την αρχή μια καινούρια κλάση για τη μηχανή που να κάνει την ίδια κίνηση με το αυτοκίνητο (ίδια μέθοδο move) αλλά τυπώνεται διαφορετικά (διαφορετική μέθοδο printPosition)

- Τι προβλήματα έχει αυτό?
  - Επανάληψη του κώδικα (μπορεί να είναι μεγάλος και δύσκολος)
  - Πιθανότητα για λάθη
  - Πολύ δύσκολο να γίνουν αλλαγές (θα πρέπει να γίνονται σε δύο σημεία)

Κληρονομικότητα

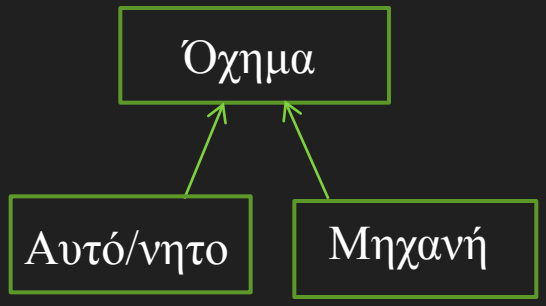
- Ορίζουμε μια κλάση Vehicle (όχημα) η οποία έχει θέση, και έχει κίνηση (μέθοδο move)




```
Πρόγραμμα
Car myCar = new Car()
Motorcycle myMoto = new Motorcycle()
myCar.printPosition()
myMoto.printPosition()
For i = 1…10
    myCar.move()
    myCar.printPosition()
    myMoto.move()
    myMoto.printPosition()
```

Κληρονομικότητα

- Οι κλάσεις μας επιτρέπουν να ορίσουμε μια ιεραρχία
  - Π.χ., και ο Φοιτητής και ο Καθηγητής ανήκουν στην κλάση Άνθρωπος.
  - Η κλάση Αυτοκίνητο ανήκει στην κλάση Όχημα η οποία περιέχει και την κλάση Μοτοσυκλέτα
- Οι κλάσεις πιο χαμηλά στην ιεραρχία κληρονομούν χαρακτηριστικά και συμπεριφορά από τις ανώτερες κλάσεις
  - Όλοι οι άνθρωποι έχουν όνομα
  - Όλα τα οχήματα έχουν μέθοδο drive, stop.



Παράδειγμα: Ιεραρχία κλάσεων

- Δίνεται η εξής περιγραφή:
  - «Υπάρχουν διάφορα είδη οχημάτων. Τρία τέτοια είδη είναι τα οχήματα εδάφους, τα οχήματα θαλάσσης και τα ιπτάμενα οχήματα. Οχήματα εδάφους είναι τα αυτοκίνητα και οι άμαξες, ενώ ιπτάμενα οχήματα είναι τα αεροπλάνα και τα ελικόπτερα. Οχήματα θαλάσσης είναι οι βάρκες και τα πλοία. Υπάρχουν δύο αεροπλάνα, τα ΑΛΕΞΑΝΔΡΟΣ και ΜΑΚΕΔΟΝΙΑ, ένα ελικόπτερο, το ΑΕΤΟΣ, μια βάρκα, η ΑΥΡΑ, καθώς και δύο πλοία, τα ΝΑΞΟΣ και ΚΡΗΤΗ».
  - (α) Σχεδιάστε μια ιεραρχία που να περιλαμβάνει όλες τις κλάσεις και τα στιγμιότυπα.
  - (β) Μας δίνεται επί πλέον η εξής πληροφορία: «Τα υδρόπτερα είναι και ιπτάμενα και θαλάσσια οχήματα». Ενημερώστε (σχεδιαστικά) την ιεραρχία, ώστε να περιλάβει τη νέα πληροφορία

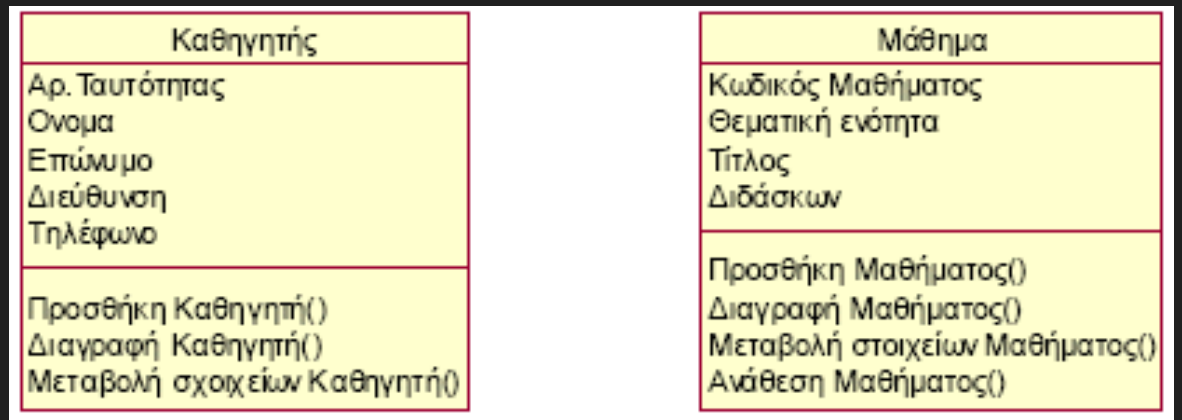
Οι κλάσεις σχηματίζουν ιεραρχία

- Οι κλάσεις δομούνται σε μια δενδρική ιεραρχία
- Η κλάση στη ρίζα της ιεραρχίας ονομάζεται Object
- Κάθε κλάση εκτός από την Object, έχει μια υπερκλάση (superclass)
- Μια κλάση μπορεί να έχει πολλούς προγόνους μέχρι την Object
- Όταν ορίζουμε μια κλάση, ορίζουμε την υπερκλάση της
  - Αν δεν ορίσουμε υπερκλάση θεωρείται η Object
- Κάθε κλάση μπορεί να έχει μία ή περισσότερες υποκλάσεις (subclasses)

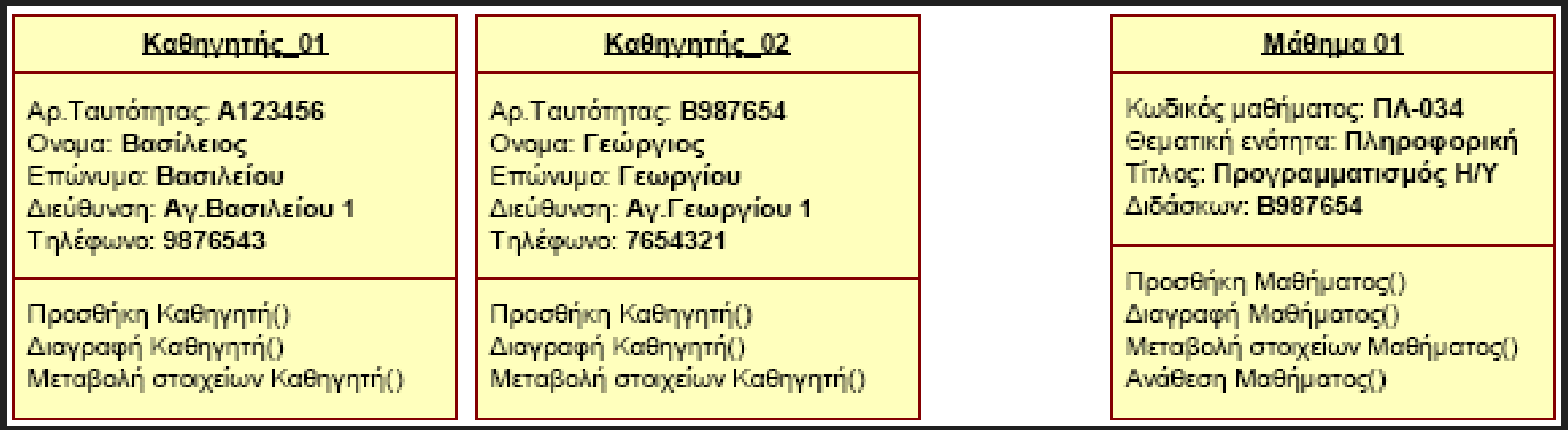
ΚΛΑΣΕΙΣ ΚΑΙ ΑΝΤΙΚΕΙΜΕΝΑ στη UML

Κλάσεις και αντικείμενα στη UML

- Κλάσεις



- Αντικείμενα



Σχέσεις μεταξύ κλάσεων

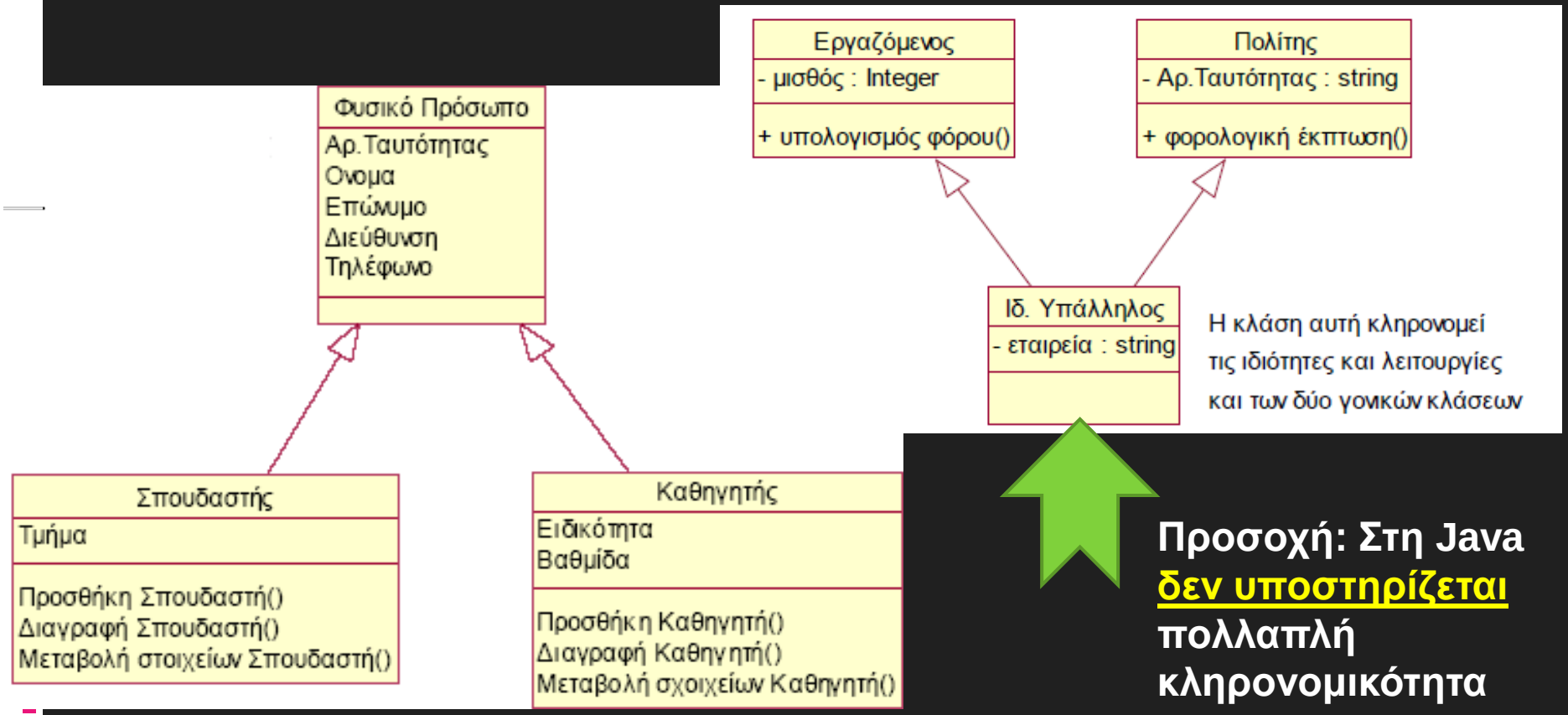
- Συσχέτιση (association): Μια γενική σχέση μεταξύ κλάσεων
  - Περιγραφή συσχέτισης (όνομα)
  - Πολλαπλότητα
  - Ρόλοι

Κληρονομικότητα (1/2)

- ***"is a"***, ***"kind of"***
- ***Κληρονομικότητα*** ή ***γενίκευση*** (inheritance, generalisation): Η απόδοση χαρακτηριστικών από μια κλάση (πατέρας) σε άλλες (παιδιά).
  - Απλή: κάθε κλάση έχει έναν «πατέρα»
  - Πολλαπλή: κάθε κλάση έχει πάνω από έναν «πατέρα»
- Κληρονομικότητα και γενίκευση αποτελούν τις δύο όψεις ενός μηχανισμού ***ταξινόμησης*** (classification) οντοτήτων του πεδίου του προβλήματος:
  - Η κλάση-παιδί είναι εξειδίκευση της κλάσης-πατέρα
  - Η κλάση-πατέρας είναι γενίκευση της κλάσης-παιδί
- Οι υποκλάσεις προσφέρουν *εξειδικευμένη* συμπεριφορά από τα κοινά στοιχεία που προσφέρει η υπερκλάση.
- Επαναχρησιμοποίηση του κώδικα της υπερκλάσης.

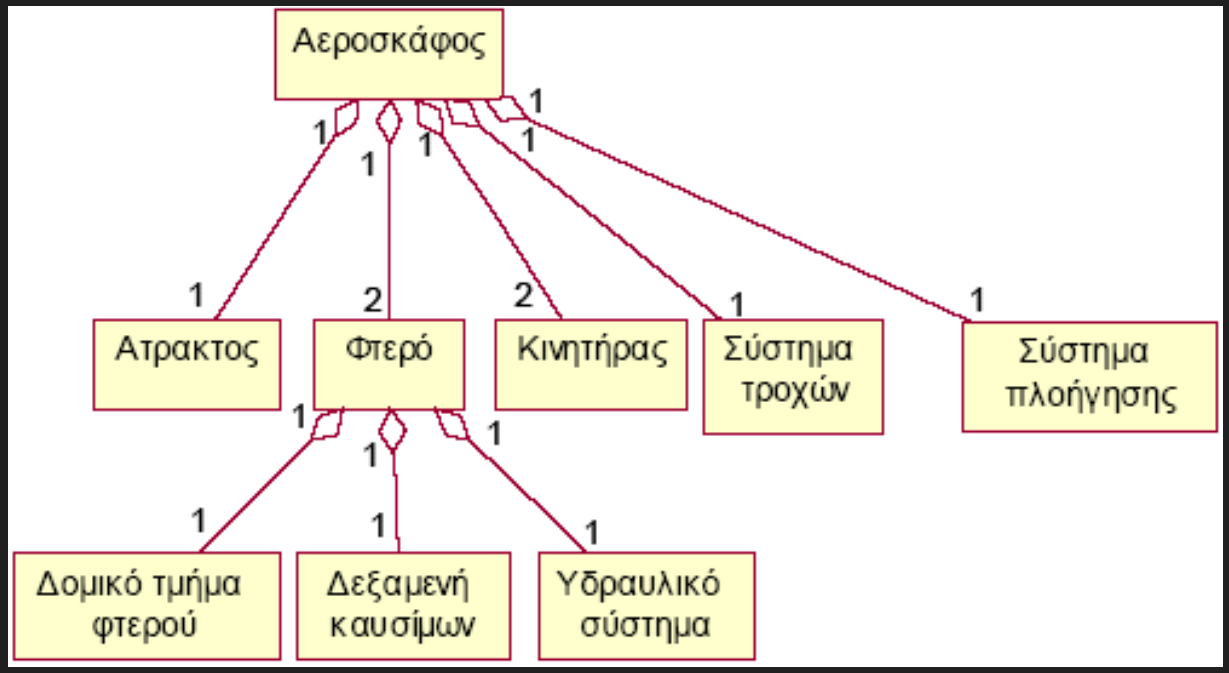
Κληρονομικότητα (2/2)

- Κάθε κλάση-παιδί έχει όλα τα χαρακτηριστικά της κλάσης-πατέρα και ορισμένα επιπλέον



Συναρμολόγηση/Συνάθροιση (aggregation)

- ***"has"***, ***"is part of"***
- Η σχέση που συνδέει κλάσεις που περιγράφουν τη σύνθεση συνόλων από απλούστερα μέρη



Εισαγωγή στη JAVA

Java (Εισαγωγή σε μία διαφάνεια…)

- Αναπτύχθηκε από τη Sun Microsystems
  - 1η έκδοση το 1995. Σήμερα έχουμε την έκδοση 8 (Μάρτιος 2014, τελευταίο update: February 5, 2016)
- Δωρεάν διάθεση από http://www.oracle.com/technetwork/java/index.html
- Το συντακτικό βασίζεται σε C / C++

- Είναι platform-independent:
  - ο πηγαίος κώδικας μεταγλωττίζεται σε bytecode (ενδιάμεση μορφή εντολών μεταξύ γλώσσας υψηλού επιπέδου και γλώσσας μηχανής)
  - ένα πρόγραμμα σε bytecode μπορεί να εκτελεστεί σε οποιονδήποτε Η/Υ έχει εγκατεστημένη τη Java Virtual Machine

Τεκμηρίωση

- Εξαιρετική τεκμηρίωση για τη Java από την Oracle:

http://docs.oracle.com/javase/tutorial/

- Για οποιαδήποτε κλάση βιβλιοθηκών της Java (Java API) μία αναζήτηση στο Google του τύπου όνομαΚλάσης Java (π.χ. String Java)… συνήθως επιστρέφει ως πρώτο αποτέλεσμα την σελίδα της Oracle με την επίσημη τεκμηρίωση
- Εκδόσεις API
  - Java SE (Standard Edition): Παρέχει όλη τη βασική λειτουργικότητα της Java για ανάπτυξη εφαρμογών, δικτύωση, ασφάλεια, πρόσβαση σε ΒΔ, GUI.
  - Java EE (Enterprise Edition): παρέχει κλάσεις βιβλιοθήκης (API) για εφαρμογές μεγάλης κλίμακας (multi-tiered, scalable, reliable, secure network applications)
  - Java ME (Micro Edition): Ανάπτυξη εφαρμογών σε φορητές συσκευές
  - JavaFX: Ανάπτυξη 'ελαφρών' διαδικτυακών εφαρμογών

Ορισμός κλάσης

- Κάθε κλάση κατά προτίμηση τοποθετείται σε ξεχωριστό αρχείο .java με όνομα το όνομα της κλάσης
- Ορατότητα ιδιοτήτων και μεθόδων
  - -: ιδιωτική ορατότητα (private)
  - +: δημόσια ορατότητα (public)

Order
-code : String
+getTotalPrice() : double

```
public class Order {

  private String code;

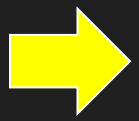
  public double getTotalPrice()
{
      //σώμα της μεθόδου
      //σχόλιο μιας γραμμής
      /* σχόλιο πολλών γραμμών
         με αυτό το σύμβολο */
}
```

Κατασκευαστές (Constructors)

- Κάθε κλάση διαθέτει μία «ειδική» μέθοδο που καλείται κατά την κατασκευή νέου αντικειμένου της κλάσης (κατασκευαστής – constructor)
- Αν δεν δηλωθεί κατασκευαστής, χρησιμοποιείται ο εξ' ορισμού κατασκευαστής που απλώς δημιουργεί το αντικείμενο χωρίς να αρχικοποιεί τις τιμές των ιδιοτήτων
- Ένας κατασκευαστής πρέπει υποχρεωτικά να έχει το όνομα της κλάσης
- Σε έναν κατασκευαστή με παραμέτρους μπορούμε να αποδώσουμε αρχικές τιμές σε ιδιότητες

Κατασκευαστές (Constructors)

Order
- code : String
+ Order(text : String)



```
public class Order {

  private String code;

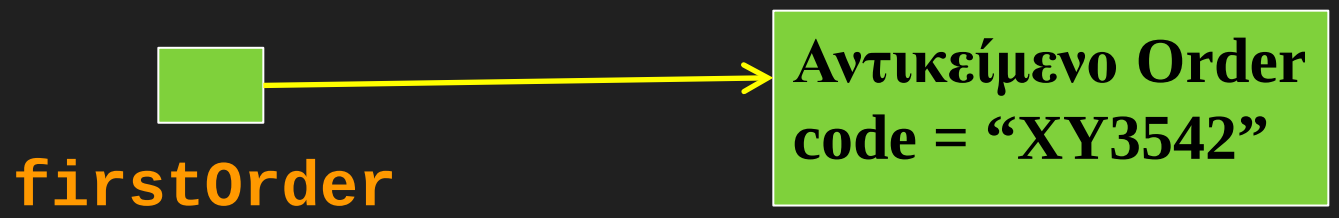

  public Order(String text) {

     code = text;

  }
}
```

Δημιουργία Αντικειμένων

- Δημιουργούμε ένα αντικείμενο κλάσης στη Java με χρήση της δεσμευμένης λέξης new και κλήση του κατασκευαστή της κλάσης
- Το νέο αντικείμενο το αναθέτουμε σε μία αναφορά (reference) του τύπου της κλάσης:
- Order firstOrder; //δημιουργία αναφοράς προς αντικείμενο τύπου Order
- firstOrder = new Order("XY3542"); //ανάθεση τιμής στην αναφορά
- Το firstOrder είναι μία αναφορά τύπου Order. Κατ' ουσίαν πρόκειται για έναν δείκτη (pointer) προς αντικείμενο τύπου Order



- Καταχρηστικά, αναφερόμαστε συνήθως στο ίδιο το αντικείμενο ως firstOrder

Δημιουργία αντικειμένων

- Στη Java, τα πάντα είναι αντικείμενα (και κατά συνέπεια τα χειριζόμαστε μέσω αναφορών σε αυτά)
  - String name = "Markos";
    το name είναι αναφορά προς αντικείμενο τύπου String
  - int[] anArray = new int[10];
    το anArray είναι αναφορά προς αντικείμενο τύπου int[], δηλ. πίνακας ακεραίων

- … εκτός από 8 Στοιχειώδεις τύπους που δεν είναι αντικείμενα:
  - byte, short, int, long, float, double, boolean, char

Βασικοί τύποι

Όνομα τύπου	Τιμή	Μνήμη	Τιμές
boolean	true/false	1 byte	true, false
char	Χαρακτήρας (Unicode)	2 bytes	Γράμματα, αριθμοί, σημεία στίξης και άλλα σύμβολα
byte	Ακέραιος	1 byte	-128 έως 127
short	Ακέραιος	2 bytes	-32.768 έως 32.767
int	Ακέραιος	4 bytes	-2.147.483.648 έως 2.147.483.647
long	Ακέραιος	8 bytes	-9,223,372,036,854,775,808 έως....
float	Πραγματικός	4 bytes	1,4E-45 έως 3,4^{E}+38
double	Πραγματικός	8 bytes	4,9E-324 έως 1,7 E+308

Όταν ορίζουμε μια μεταβλητή δεσμεύεται ο αντίστοιχος χώρος στη μνήμη. Το όνομα της μεταβλητής αντιστοιχίζεται με αυτό το χώρο στη μνήμη.

Παράδειγμα ορισμού κλάσης

```
class Employee {
    // Πεδία ή ιδιότητες κλάσεις
    private String name;     // Σύμφωνα με τις μεθόδους που ορίζονται
στη συνέχεια, διαβάζεται αλλά δεν αλλάζει
    private double salary;  // Σύμφωνα με τις μεθόδους που ορίζονται
στη συνέχεια, δεν διαβάζεται, δεν αλλάζει
    // Κατασκευαστής
    Employee (String n, double s) {
        name = n;
        salary = s;
    }
    // Μέθοδοι
    void pay () {
        System.out.println("Pay to the order of " +
                                    name + " $" + salary);
    }
    public String getName() { return name; } // getter
}
```

Δημιουργία αντικειμένων

- Όλα τα προγράμματα Java πρέπει να περιλαμβάνουν μία μέθοδο main από την οποία ξεκινά η εκτέλεση του προγράμματος
- Στη main μπορούμε να δημιουργήσουμε αντικείμενα των υπόλοιπων κλάσεων
- Η μέθοδος main μπορεί να βρίσκεται σε οποιαδήποτε κλάση. Συνήθως την τοποθετούμε μέσα σε μια κλάση με όνομα Main

```
public class Main{

    public static void main(String[] args) {

        //Δημιουργία αντικειμένων

    }
}
```

HELLO WORLD!

File HelloWorld.java

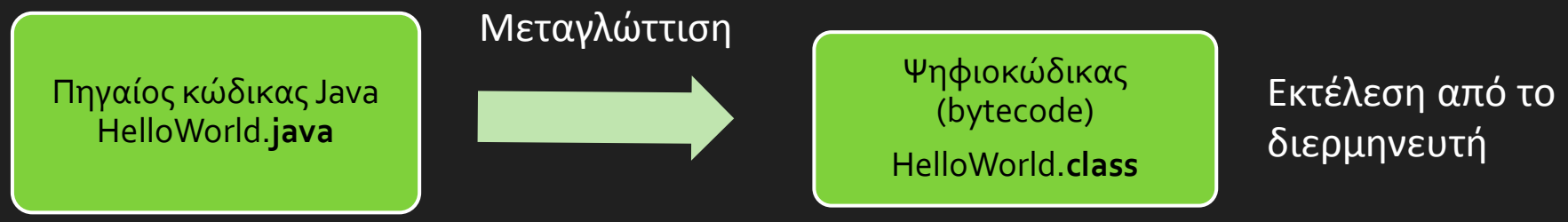
```
class HelloWorld
{
    public static void main(String args[])
    {
            // print message
            System.out.println("Hello world!");
      }
}
```

- javac HelloWorld.java
- java HelloWorld

Χωρίς καμία κατάληξη!

.java και .class

- Η Java είναι αρχιτεκτονικά ουδέτερη γλώσσα (Platform independent)
- Ένα αρχείο .class μπορεί να χρησιμοποιηθεί σε οποιοδήποτε υπολογιστή έχει εγκατεστημένη τη JVM
- Τα αρχεία .class έχουν πολύ μικρό μέγεθος
- Διερμηνευτής: JVM



Ορίζει την κλάση

Όνομα της κλάσης

```
class HelloWorld
{
    public static void main(String args[])
    {
        // print message
        System.out.println("Hello world!");
    }
}
```

```
class HelloWorld
{
    public static void main(String args[])
    {
        // print message
        System.out.println("Hello world!");
    }
}
```

Τα άγκιστρα { … } ορίζουν ένα λογικό block του κώδικα

- Αυτό μπορεί να είναι μία κλάση, μία συνάρτηση, ένα if statement
- Οι μεταβλητές που ορίζουμε μέσα σε ένα λογικό block, έχουν εμβέλεια μέσα στο block

Ορισμός της μεθόδου main

```
class HelloWorld
{
    public static void main(String args[])
    {
            // print message
            System.out.println("Hello world!");
    }
}
```

public, static: θα τα εξηγήσουμε σε επόμενο μάθημα
void: Η μέθοδος δεν επιστρέφει τίποτα
main: σηματοδοτεί το σημείο εκκίνησης του προγράμματος

Ορισμός της μεθόδου main

```
/**
* A class that prints a message “hello world”
*/
class HelloWorld
{
    public static void main(String args[])
    {
            // print message
            System.out.println(“Hello world!”);
        }
}
```

Ορίσματα της μεθόδου

- Ένας πίνακας από Strings που αντιστοιχούν στις παραμέτρους με τις οποίες τρέχουμε το πρόγραμμα.
- String: κλάση βιβλιοθήκης της Java που χειρίζεται τα αλφαριθμητικά

Σχόλια

```
/**
A class that prints a message “hello world”
*/
class HelloWorld
/* σχόλιο
Πολλών γραμμών*/

{
    public static void main(String args[])
    {
            // σχόλιο 1 γραμμής
            System.out.println(“Hello world!”);
        }
}
```

Κάθε εντολή στη Java πρέπει να τερματίζει με το ;

```
class HelloWorld
{
    public static void main(String args[])
    {
        // print message
        System.out.println("Hello world!");
    }
}
```

Αντικείμενο **System.out**

Μέθοδος println:
Τυπώνει το String που δίνεται ως όρισμα και αλλάζει γραμμή

http://www.programmerinterview.com/index.php/java-questions/how-system-out-println-works/

Παράδειγμα

- Φτιάξτε ένα πρόγραμμα που τυπώνει το αποτέλεσμα της διαίρεσης δύο ακεραίων.

Division.java

```
class Division
{
  public static void main(String args[])
  {
    int enumerator = 32;
    int denominator = 10;
    double division;
    division = enumerator/(double)denominator;
    System.out.println(“Result = “ + division);
  }
}
```

Division.java

```
class Division
{
  public static void main(String args[])
  {
    int enumerator = 32;
    int denominator = 10;
    double division;
    division = enumerator/(double)denominator;
    System.out.println("Result = " + division);
  }
}
```

Ορισμός μεταβλητών

- Η Java είναι strongly typed γλώσσα: κάθε μεταβλητή θα πρέπει να έχει ένα τύπο
- Οι τύποι int και double είναι βασικοί τύποι (primitive types)
- Εκτός από τους βασικούς τύπους, όλοι οι άλλοι τύποι είναι κλάσεις

Division.java

```
class Division
{
  public static void main(String args[])
  {
    int enumerator = 32;
    int denominator = 10;
    double division;
    division = enumerator/(double)denominator;
    System.out.println(“Result = “ + division);
  }
}
```

Ανάθεση: αποτίμηση της τιμής της έκφρασης στο δεξιό μέλος του “=” και μετά ανάθεση της τιμής στην μεταβλητή στο αριστερό μέλος

Division.java

```
class Division
{
  public static void main(String args[])
  {
    int enumerator = 32;
    int denominator = 10;
    double division;
    division = enumerator/(double)denominator;
    System.out.println("Result = " + division);
  }
}
```

Μετατροπή τύπου: `(double)denominator` μετατρέπει την τιμή της μεταβλητής `denominator` σε `double`.
Αν δεν γίνει η μετατροπή, η διαίρεση μεταξύ ακεραίων μας δίνει πάντα ακέραιο.

Αναθέσεις

- Στην ανάθεση κατά κανόνα, η τιμή του δεξιού μέρους θα πρέπει να είναι ίδιου τύπου με την μεταβλητή του αριστερού μέρους.
- Υπάρχουν εξαιρέσεις όταν υπάρχει συμβατότητα μεταξύ τύπων

- byte → short → int → long → float → double
  - Μια τιμή τύπου T μπορούμε να την αναθέσουμε σε μια μεταβλητή τύπου που εμφανίζεται δεξιά του T

- (Σε αντίθεση με την C) ο τύπος boolean δεν είναι συμβατός με τους ακέραιους

Division.java

```
class Division
{
  public static void main(String args[])
  {
    int enumerator = 32;
    int denominator = 10;
    double division;
    division = enumerator/(double)denominator;
    System.out.println(“Result = “ + division);
  }
}
```

Ο τελεστής “+” μεταξύ αντικείμενων της κλάσης String συνενώνει (concatenates) τα δύο String.
Μεταξύ ενός String και ενός βασικού τύπου, ο βασικός τύπος μετατρέπεται σε String και γίνεται η συνένωση

Αλφαριθμητικά (strings)

- Η κλάση String είναι προκαθορισμένη κλάση της Java που μας επιτρέπει να χειριζόμαστε αλφαριθμητικά.
- Ο τελεστής "+" μας επιτρέπει την συνένωση
- Υπάρχουν πολλές χρήσιμες μέθοδοι της κλάσης String.
  - length(): μήκος του String
  - equals(String x): ελέγχει για ισότητα του αντικειμένου που κάλεσε την μέθοδο και του ορίσματος x.
  - trim(): αφαιρεί κενά στην αρχή και το τέλος του string.
  - Μέθοδοι για να βρεθεί ένα υπο-string μέσα σε ένα string.
  - κ.α.

Έξοδος στην οθόνη

- Μπορούμε να καλέσουμε τις μεθόδους του System.out:
  - println(String s): για να τυπώσουμε ένα αλφαριθμητικό s και τον χαρακτήρα '\n' (αλλαγή γραμμής)
  - print(String s): τυπώνει το s αλλά δεν αλλάζει γραμμή
  - printf: Formatted output
    - printf("%d",myInt); // τυπώνει ένα ακέραιο
    - printf("%f",myDouble); // τυπώνει ένα πραγματικό
    - printf("%.2f",myDouble); // τυπώνει ένα πραγματικό με δύο δεκαδικά

Είσοδος από το πληκτρολόγιο

- Χρησιμοποιούμε την κλάση Scanner της Java
  - import java.util.Scanner;
- Αρχικοποιείται με το ρεύμα εισόδου:
  - Scanner in = new Scanner(System.in);
- Μπορούμε να καλέσουμε μεθόδους της Scanner για να διαβάσουμε κάτι από την είσοδο
  - nextLine(): διαβάζει μέχρι να βρει τον χαρακτήρα '\n'
  - next(): διαβάζει το επόμενο String
  - nextInt(): διαβάζει τον επόμενο int
  - nextDouble(): διαβάζει τον επόμενο double.

Παράδειγμα

```
import java.util.Scanner;

class TestIO
{
	public static void main(String args[])
	{
		Scanner input = new Scanner(System.in);
		String line = input.nextLine();
		System.out.println(line);
	}
}
```

new: δημιουργεί ένα αντικείμενο τύπου Scanner (μία μεταβλητή) με το οποίο μπορούμε πλέον να διαβάζουμε από την είσοδο

Division.java με είσοδο από το χρήστη

```
class Division
{
  public static void main(String args[])
  {
// Δημιουργεί το Scanner για να πάρει είσοδο από την κονσόλα
        Scanner input = new Scanner(System.in)
        int enumerator;
        int denominator;
        double division;
        System.out.println("Enter first integer:");
        enumerator = input.nextInt();
        System.out.println("Enter second integer:");
        denominator = input.nextInt();

        division = enumerator/(double)denominator;
        System.out.println("Result = " + division);
  }
}
```

Μέχρι στιγμής τίποτα αντικειμενοστραφές!

- Δεν κατασκευάσαμε κλάσεις και αντικείμενα
- Αν και χρησιμοποιήσαμε κάποιες κλάσεις βιβλιοθήκης της Java (String, Scanner) και κάποιες 'έτοιμες' μεθόδους
- Θα δούμε ένα παράδειγμα κώδικα σε δύο εκδοχές
  - μια μη αντικειμενοστρεφή
  - μια αντικειμενοστρεφή

Παράδειγμα

- Ζητούμενη λειτουργικότητα
  - Δίνουμε σαν είσοδο χαρακτηριστικά προϊόντων
    - Όνομα
    - Τιμή
    - Σκορ
  - Αυτό γίνεται μέχρι να δηλώσουμε ότι δεν έχουμε άλλο προϊόν να εισάγουμε
  - Το πρόγραμμα υπολογίζει από όλα τα προϊόντα το καλύτερο (αυτό με τον μεγαλύτερο λόγο σκορ/τιμή)
  - Εμφανίζει τα στοιχεία του προϊόντος αυτού

Οντοκεντρικός Προγραμματισμός

ΦΡΟΝΤΙΣΤΗΡΙΟ JAVA

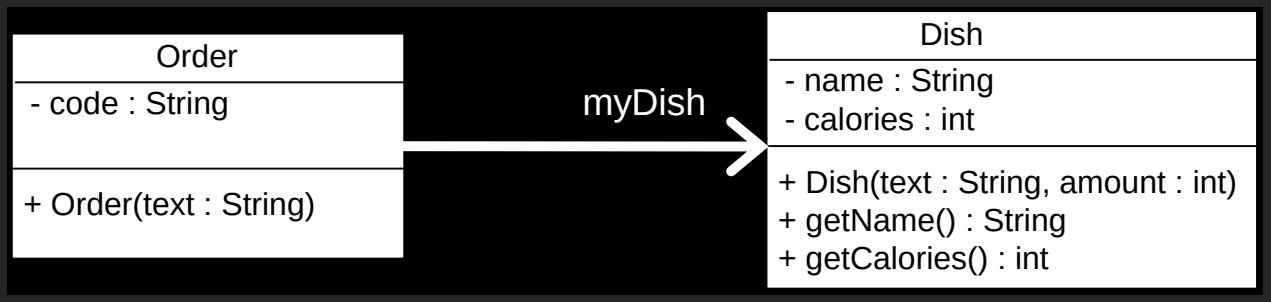
Τι θα συζητήσουμε σήμερα

- Πώς υλοποιούμε συσχετίσεις μεταξύ κλάσεων
  - απλές και πολλαπλές συσχετίσεις
  - κληρονομικότητα
- Static, final
- Overloading – Overriding – Hiding
- Παραδείγματα κώδικα

Συσχετίσεις μεταξύ κλάσεων

- Οι συσχετίσεις είναι η «κόλλα» ενός αντικειμενοστρεφούς συστήματος.
- Παρέχουν την υποδομή για την επικοινωνία μεταξύ αντικειμένων
- Υλοποιούνται ως αναφορές από μία κλάση προς μία άλλη

Συσχετίσεις μεταξύ κλάσεων




```
public class Order {

  private Dish myDish;    //αναφορά προς την κλάση Dish
                          //υλοποιεί τη συσχέτιση

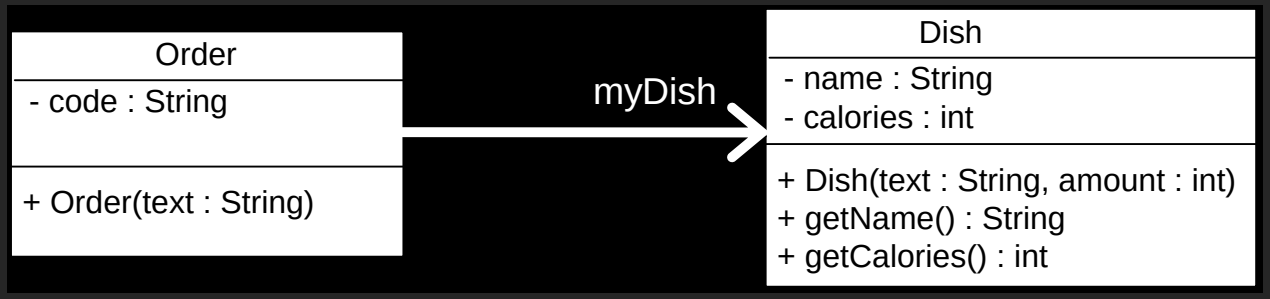
  private String code;


  public Order(String text) {
     code = text;
  }
}
```

Συσχετίσεις μεταξύ κλάσεων

- Μία συσχέτιση παρέχει την «υποδομή» ώστε αντικείμενα δύο κλάσεων να μπορούν να επικοινωνήσουν (να ανταλλάξουν μηνύματα)
- Κατά τη διάρκεια της εκτέλεσης δημιουργούνται συνδέσεις (links) μεταξύ αντικειμένων που μπορούν να τροποποιηθούν ή να καταστραφούν
- Εάν σε μία αναφορά (που υλοποιεί μία συσχέτιση) δεν δοθεί τιμή κατά τη διάρκεια εκτέλεσης, οποιαδήποτε κλήση μεθόδου μέσω αυτής της αναφοράς είναι άνευ νοήματος (πρόκειται για pointer που δεν «δείχνει» πουθενά)
  - Οδηγεί σε σφάλμα Null Pointer Exception

Συσχετίσεις μεταξύ κλάσεων - Συνδέσεις



Αντικείμενο o1 τύπου Order

Αντικείμενο d1 τύπου Dish

- "Συνδέουμε" το αντικείμενο o1 με το αντικείμενο d1, δίνοντας στην ιδιότητα myDish του αντικειμένου o1 την τιμή d1
- Ενδεχομένως να χρειαστεί κατάλληλη μέθοδος set( ) στην κλάση Order

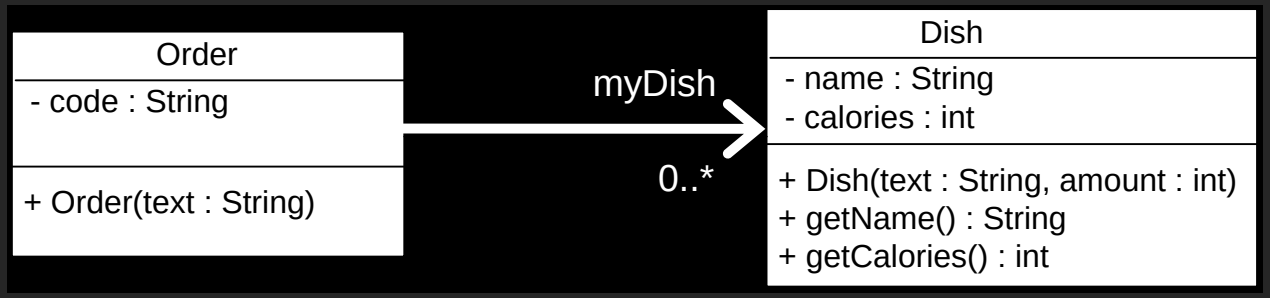
Συσχετίσεις μεταξύ κλάσεων - Συνδέσεις


```
public class Order {
  private Dish myDish;   //αναφορά προς την κλάση Dish

  public void setDish(Dish aDish) {
     myDish = aDish;
  }
}
```

```
public static void main(String[] args) {
    Order o1 = new Order("XY3542");
    Dish d1 = new Dish("Arakas");
    o1.setDish(d1);  //εδώ δημιουργείται η σύνδεση
                     //το αντικείμενο o1 μπορεί πλέον
                     //να αποστείλει μηνύματα στο d1
}
```

Συσχετίσεις με πολλαπλότητα



- Η πολλαπλότητα " 0..* " στο άκρο της κλάσης Dish, υποδηλώνει ότι ένα αντικείμενο τύπου Order μπορεί να συσχετιστεί με πολλά αντικείμενα τύπου Dish
- Υλοποιείται ως μία δομή δεδομένων στην κλάση Order που περιλαμβάνει αναφορές προς αντικείμενα τύπου Dish
- Η πιο ευρέως διαδεδομένη και εύκολη στη χρήση δομή δεδομένων της Java είναι η ArrayList

ArrayList

- Για να χρησιμοποιηθεί πρέπει πρώτα να συμπεριληφθεί το πακέτο (κατάλογος) στο οποίο είναι δηλωμένη
- import java.util.*; (συμπερίληψη όλων των κλάσεων του πακέτου)
- Αποτελεί ουσιαστικά έναν δυναμικό πίνακα που μπορεί να φιλοξενήσει στοιχεία οποιουδήποτε τύπου
- Δημιουργία αντικειμένου ArrayList

```
ArrayList dishes = new ArrayList();
```

- Από την Java 1.5 μπορούμε να δηλώσουμε τον τύπο των στοιχείων που φιλοξενούνται στη δομή (π.χ. αντικείμενα Dish) ως εξής:

```
ArrayList<Dish> dishes = new ArrayList<Dish>();
```

ArrayList

- Εισαγωγή στοιχείου στη δομή καλώντας τη μέθοδο add():

```
Dish d1 = new Dish("Steak");
dishes.add(d1);
```

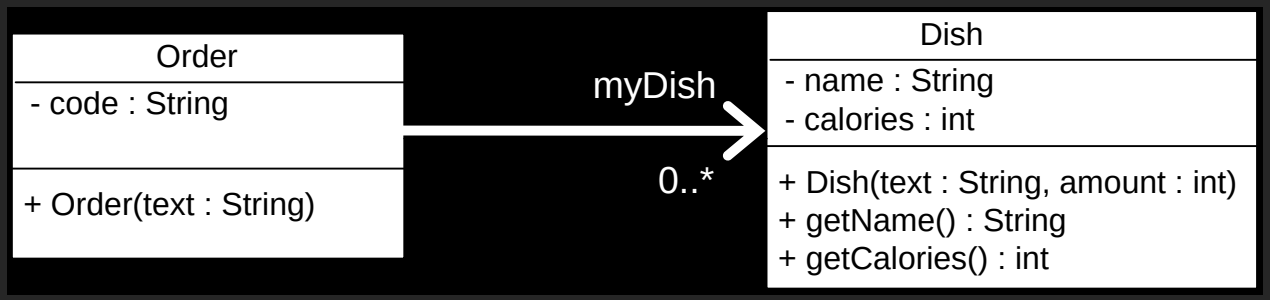
- Λήψη του στοιχείου που βρίσκεται στη θέση i με τη μέθοδο get():

```
Dish d = dishes.get(i);
```

- Διαγραφή του στοιχείου d1 από τη δομή:

```
dishes.remove(d1);
```

Συσχετίσεις με πολλαπλότητα

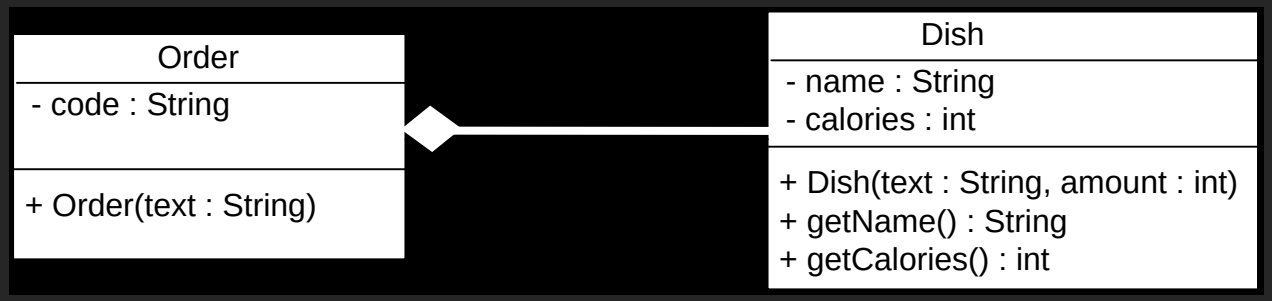



```
public class Order {
  private ArrayList<Dish> myDishes =
                                  new ArrayList<Dish>();

  public void addDish(Dish aDish) {
     myDishes.add(aDish);
  }

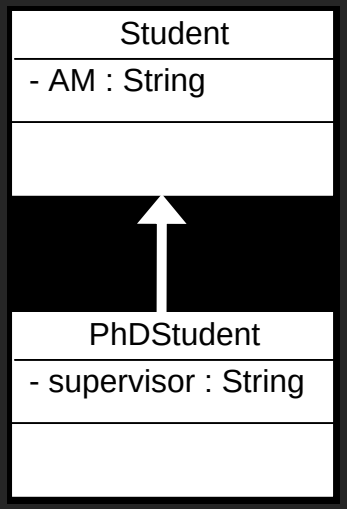
  public Dish getDish(int i) {
     myDishes.get(i);
  }
}
```

Σχέσεις Περιεκτικότητας



- Μια σχέση περιεκτικότητας (π.χ. συσσωμάτωση) υλοποιείται στη Java ακριβώς όπως και μια απλή συσχέτιση

Κληρονομικότητα

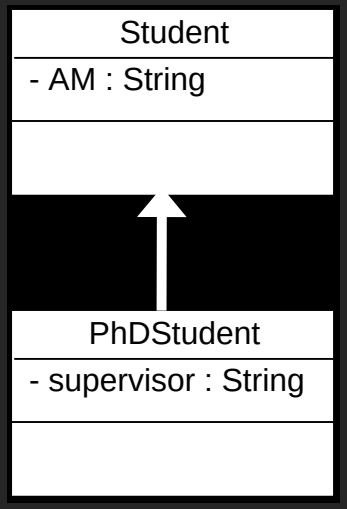



```
public class PhDStudent extends Student {

    private String supervisor;

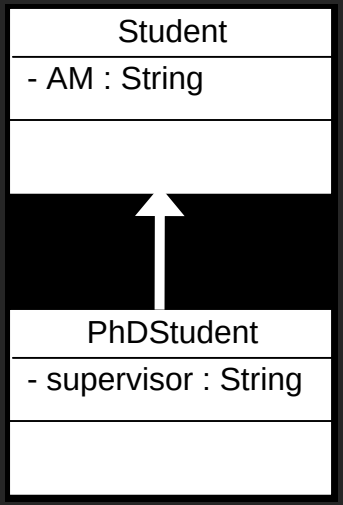
}
```

Κληρονομικότητα



- Η υποκλάση (PhDStudent) κληρονομεί τις ιδιότητες και τις μεθόδους της υπερκλάσης (Student)
  - (ο κατασκευαστής δεν κληρονομείται)
- Η υποκλάση μπορεί να δηλώσει επιπλέον μεθόδους ή ιδιότητες
- Η υποκλάση μπορεί να επαναορίσει (επικαλύψει) μεθόδους που έχουν οριστεί στην υπερκλάση

Αρχή της Υποκατάστασης



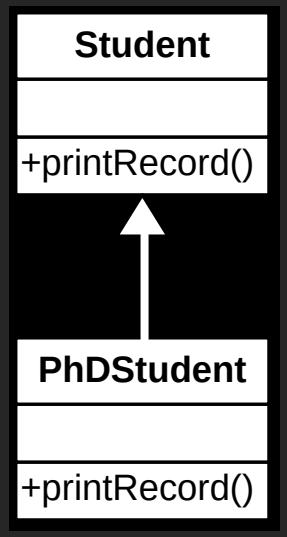
Σε μία αναφορά προς την υπερκλάση είναι απολύτως έγκυρο να αναθέσουμε ως τιμή ένα αντικείμενο οποιασδήποτε υποκλάσης

```
...
   Student p;

   p = new Student();      //επιτρεπτό

   p = new PhDStudent();   //επιτρεπτό
```

Πολυμορφισμός



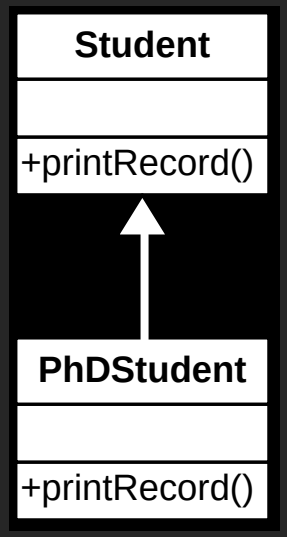
Το γεγονός ότι η μέθοδος printRecord() σημειώνεται και στην κλάση PhDStudent
υποδηλώνει ότι η μέθοδος επαναορίζεται στην υποκλάση

```
. . .
   Student p;

   p = new PhDStudent();

   p.printRecord(); //παρόλο που η αναφορά p είναι
                    //δηλωμένη ως αναφορά τύπου Student
                    //η μέθοδος που καλείται είναι η
                    //printRecord() της PhDStudent
```

Πολυμορφισμός



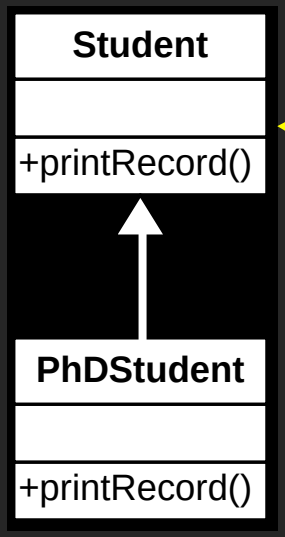
Το γεγονός ότι η μέθοδος printRecord() σημειώνεται και στην κλάση PhDStudent
υποδηλώνει ότι η μέθοδος επαναορίζεται στην υποκλάση

```
. . .
   Student p;

   p = new PhDStudent();

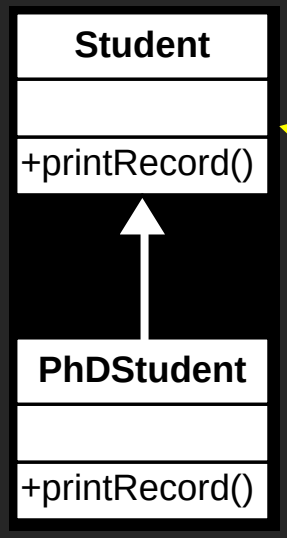
   p.printRecord();//η Java αναμένει μέχρι την εκτέλεση
                   //του προγράμματος για να δεί που
                   //"δείχνει" η αναφορά p
                 // = δυναμική ή καθυστερημένη διασύνδεση
                 // = ΠΟΛΥΜΟΡΦΙΣΜΟΣ
```

Πολυμορφισμός



- Ο πολυμορφισμός είναι εξαιρετικά σημαντικός για τη συντήρηση του λογισμικού διότι:
- ένα πρόγραμμα πελάτης έχει γραφεί ώστε να φαίνεται ότι χρησιμοποιεί αντικείμενα τύπου Student (και κατά συνέπεια καλεί τη μέθοδο printRecord)
- αλλά το ίδιο πρόγραμμα πελάτης μπορεί να λειτουργήσει με οποιοδήποτε αντικείμενο υποκλάσης της Student (ακόμα και κλάσεων που δεν υπάρχουν ακόμα αλλά θα δημιουργηθούν στο μέλλον)
- ΧΩΡΙΣ ΚΑΜΜΙΑ ΑΠΟΛΥΤΩΣ ΑΛΛΑΓΗ ΣΤΟΝ ΚΩΔΙΚΑ ΤΟΥ ΠΕΛΑΤΗ

Πολυμορφισμός




```
public static void main (String[ ] args)
{
ArrayList<Student> students = new ArrayList<Student>();
students.add(new Student("John", "5231"));
students.add(new PhDStudent("Sara",
"541","TomeasYlikou"));

For (int i=0; i<students.size(); i++)
students.get(i).printRecord(); //Πολυμορφική κλήση
}
```

Στατικά μέλη κλάσεων

- Όλα τα αντικείμενα μιας κλάσης έχουν τις ίδιες ιδιότητες αλλά κάθε αντικείμενο έχει διαφορετική τιμή για μια ιδιότητα
- Υπάρχουν περιπτώσεις όπου όλα τα αντικείμενα μιας κλάσης θέλουμε να μοιράζονται την ίδια τιμή για μία ιδιότητα
- Π.χ. ιδιότητα "πλήθος Υπαλλήλων" για μια κλάση Υπάλληλος
- Θα ήταν πλεονασμός κάθε αντικείμενο τύπου Υπάλληλος να έχει την ίδια τιμή για το "πλήθος Υπαλλήλων" και επιπλέον κάθε φορά που προστίθεται ένας νέος υπάλληλος να πρέπει να ενημερωθούν όλα τα αντικείμενα

Στατικά μέλη κλάσεων

- Στις περιπτώσεις αυτές αξιοποιούμε τις στατικές ιδιότητες
- Μία στατική ιδιότητα:
  - υφίσταται σε επίπεδο κλάσης (ίδια τιμή για όλα τα αντικείμενα)
  - υπάρχει ακόμα και όταν δεν έχουν δημιουργηθεί αντικείμενα της κλάσης (π.χ. πλήθοςΥπαλλήλων = 0)
- Δηλώνουμε ότι μια ιδιότητα είναι στατική με το προσδιοριστικό static
- Για να μπορεί μια μέθοδος να προσπελάσει μια στατική ιδιότητα θα πρέπει να είναι και η μέθοδος στατική
  - που σημαίνει ότι η μέθοδος μπορεί να κληθεί και επί της κλάσης (χωρίς να έχουμε στα χέρια μας αντικείμενο της)

Στατικά μέλη κλάσεων

```
class Employee {

   private static int count = 0;

   public Employee() {
      count++;
   }

   public static int getCount () {
      return count;
   }
}

…
  System.out.println(Employee.getCount());  // = 0
  Employee E1 = new Employee();
  System.out.println(E1.getCount());        // = 1
```

Overloading vs. Overriding

- OVERLOADING: Όταν έχουμε μεθόδους στην ίδια κλάση με το ίδιο όνομα αλλά με διαφορετικές υπογραφές (διαφορετικό πλήθος, τύποι ή σειρά ορισμάτων)
- Η υπογραφή μίας μεθόδου είναι το όνομα της και η λίστα με τους τύπους των ορισμάτων της μεθόδου
  - Η Java μπορεί να ξεχωρίσει μεθόδους με διαφορετική υπογραφή
  - Η υπερφόρτωση γίνεται μόνο ως προς τα ορίσματα, ΌΧΙ ως προς την επιστρεφόμενη τιμή
- OVERRIDING: Όταν έχουμε σε διαφορετικές κλάσεις (σε κλάση και υποκλάση) μεθόδους με το ίδιο όνομα την ίδια υπογραφή και επιστρεφόμενους τύπους.
  - Στην υποκλάση ισχύει η μέθοδος στιγμιοτύπου που ορίστηκε τοπικά

Hiding

- Αν μια υποκλάση ορίζει μια στατική μέθοδο που έχει ίδιο όνομα και υπογραφή με μια στατική μέθοδο της υπερκλάσης της, τότε η μέθοδος στην υποκλάση αποκρύπτει τη μέθοδο της υπερκλάσης
- Όταν πρόκειται για μεθόδους κλάσεων (δηλαδή στατικές), το ποια μέθοδος θα ενεργοποιηθεί εξαρτάται από το αν καλείται από την κλάση ή την υπερκλάση
  - Όταν η στατική μέθοδος καλείται επί αντικειμένου το ποια μέθοδος θα εφαρμοστεί εξαρτάται από το πώς έχει δηλωθεί το αντικείμενο (σε ποια κλάση ανήκει – early binding).

Παράδειγμα overriding-hiding

```
class Foo {
    public static void classMethod() {
        System.out.println("classMethod() in Foo");
    }

    public void instanceMethod() {
        System.out.println("instanceMethod() in Foo");
    }
}

class Bar extends Foo {
    public static void classMethod() {
        System.out.println("classMethod() in Bar");
    }

    public void instanceMethod() {
        System.out.println("instanceMethod() in Bar");
    }
}
```

```
class Test {
    public static void main(String[] args) {
        Foo f = new Bar();
        f.instanceMethod();
        f.classMethod();
    }
}
```

Εκτελώντας παίρνουμε το εξής στην οθόνη:

```
instanceMethod() in Bar
classMethod() in Foo
```

Επεξήγηση

- Επειδή η instanceMethod() είναι μέθοδος στιγμιοτύπου, καλείται επί του αντικειμένου f που είναι τύπου Bar και άρα επικαλύπτει την αντίστοιχη μέθοδο στιγμιοτύπου στην κλάση Foo. Αν και δηλώσαμε το f σαν τύπου Foo το αντικείμενο που δημιουργήσαμε ήταν τύπου Bar (πολυμορφισμός ή καθυστερημένη διασύνδεση – late binding).
- Επειδή η μέθοδος classMethod() είναι μέθοδος κλάσης ο compiler δεν θεωρεί ότι χρειάζεται να περιμένει κατά το χρόνο εκτέλεσης ώστε να δημιουργηθούν στιγμιότυπα. Ελέγχει απλά τον τύπο που δηλώνεται κάθε αντικείμενο και αποφασίζει με βάση αυτό ποια μέθοδο θα εφαρμόσει. Εφόσον το f δηλώθηκε σαν τύπου Foo, ο compiler θεωρεί ότι το f.classMethod() σημαίνει Foo.classMethod

Method με την ίδια υπογραφή σε κλάση και υπερκλάση

	Superclass Instance Method	**Superclass Static Method**
Subclass Instance Method	Overrides	Generates a compile-time error
Subclass **Static** Method	Generates a compile-time error	Hides

http://beginnersbook.com/2013/04/oops-concepts/

Final μεταβλητές

- Δε μπορούμε να αλλάξουμε την τιμή της (είναι ουσιαστικά σταθερά)
- Μια final μεταβλητή που δεν έχει αρχική τιμή όταν ορίζεται λέγεται blank. Μπορεί να αρχικοποιηθεί μόνο μέσα σε ένα constructor.

```
class Bike{
 final int speedlimit=90;//final variable
 void run(){
  speedlimit=400;
 }
 public static void main(String args[]){
 Bike obj=new  Bike();
 obj.run();
 }
}//end of class
```

Compile Time error

Final μέθοδοι

- Όταν μια μέθοδος ορίζεται final δε μπορεί να επικαλυφθεί (να γίνει overriden)

```
class Bike{
  final void run(){System.out.println("running");}
}

class Honda extends Bike{
   void run(){System.out.println("running safely with 100kmph");}

   public static void main(String args[]){
   Honda honda= new Honda();
   honda.run();
   }
}
```

Compile Time error

Final κλάσεις

- Όταν μια κλάση ορίζεται σαν final δε μπορούμε να κατασκευάσουμε απογόνους της

```
final class Bike{}

class Honda extends Bike{
  void run(){System.out.println("running safely with 100kmph");}

  public static void main(String args[]){
  Honda honda= new Honda();
  honda.run();
  }
}
```

Compile Time error

Abstract classes, Interfaces

ΦΡΟΝΤΙΣΤΗΡΙΟ JAVA

Τι θα συζητήσουμε σήμερα

- Αφαιρέσεις στη Java
  - Abstract μέθοδοι και abstract κλάσεις
  - Interfaces (=διασυνδέσεις, διεπαφές)
  - Instanceof
  - Παραδείγματα κώδικα

Αφηρημένες μέθοδοι

- Στη Java μπορούμε να *δηλώσουμε* ένα αντικείμενο χωρίς να το *ορίσουμε*:

```
Person p;
```

- Ανάλογα, μπορούμε να *δηλώσουμε* μια μέθοδο, χωρίς να την *ορίσουμε*:

```
public abstract void draw(int size);
```

  - Υπάρχει η υπογραφή και ο επιστρεφόμενος τύπος αλλά λείπει το σώμα (η υλοποίηση) της μεθόδου
- Μια μέθοδος που έχει δηλωθεί αλλά δεν έχει οριστεί ονομάζεται abstract method

Αφηρημένες κλάσεις I

- Οποιαδήποτε κλάση περιλαμβάνει μια αφηρημένη μέθοδο είναι και αυτή αφηρημένη (abstract class)
- Πρέπει να ορίζουμε την κλάση με το keyword abstract:

```
abstract class MyClass {...}
```

- Μια αφηρημένη κλάση είναι *ημιτελής*
  - Της λείπουν κάποια από τα σώματα των μεθόδων που περιέχει
- Δε μπορούμε να δημιουργήσουμε στιγμιότυπα (αντικείμενα) από αφηρημένες κλάσεις

Αφηρημένες κλάσεις ΙΙ

- Μπορούμε να κάνουμε extend (δηλ. να ορίσουμε υποκλάσεις) σε μια abstract class
  - Αν η υποκλάση ορίζει το σώμα όλων των αφηρημένων μεθόδων που κληρονόμησε είναι «πλήρης» (concrete) και μπορεί να παράγει αντικείμενα
  - Αν ΟΧΙ, τότε και η υποκλάση θα πρέπει να δηλωθεί ως αφηρημένη (abstract)
- Μπορούμε να ορίσουμε μια κλάση ως abstract ακόμα κι αν *δεν περιέχει κάποια abstract μέθοδο*
  - Έτσι εμποδίζουμε τη δημιουργία αντικειμένων από την κλάση αυτή

Γιατί χρειαζόμαστε abstract classes?

- Έστω ότι θέλουμε να δημιουργήσουμε μια κλάση Shape, με υποκλάσεις τις Oval, Rectangle, Triangle, Hexagon, κτλ.
- Και δε θέλουμε να επιτρέπεται η δημιουργία αντικειμένων τύπου Shape
  - Έχει νόημα να υπάρχουν μόνο αντικείμενα συγκεριμένων σχημάτων
  - Αν η Shape είναι abstract, δε μπορεί να δημιουργηθεί ένα αντικείμενο Shape
  - Μπορεί όμως να δημιουργηθεί ένα αντικείμενο Oval, ή Rectangle, κτλ.
- Οι abstract κλάσεις είναι καλές για να ορίζουν μια γενική κατηγορία που περιέχει συγκεκριμένες υποκλάσεις

Παράδειγμα abstract class

- public abstract class Animal {
  abstract int eat();
  abstract void breathe();
  }
- Δε μπορούμε να δημιουργήσουμε αντικείμενα της κλάσης Animal
- Οποιαδήποτε μη αφηρημένη υποκλάση της Animal πρέπει να παρέχει υλοποιήσεις των μεθόδων eat() και breathe()

Γιατί υπάρχουν abstract methods?

- Έστω η μη αφηρημένη κλάση Shape
  - Η Shape *δεν πρέπει* να έχει τη μέθοδο draw()
  - Κάθε υποκλάση της Shape *πρέπει* να έχει μέθοδο draw() method
- Τώρα ας υποθέσουμε ότι έχουμε μια μεταβλητή figure που παίρνει σαν τιμή αντικείμενα τύπου Shape και ότι περιέχει κάποιο αντικείμενο υποκλάσης της Shape (π.χ. της κλάσης Star), δηλ.
  - Shape figure = new Star();
  - Αν γράψουμε figure.draw(), θα προκληθει syntax error επειδή ο compiler δε μπορεί να γνωρίζει τι είδους τιμή θα δοθεί στην μεταβλητή figure
  - Μια κλάση γνωρίζει την superclass στην οποία ανήκει αλλά δεν γνωρίζει τις υποκλάσεις της

Γιατί υπάρχουν abstract methods?

- **Λύση:** Ορίζουμε στην Shape μια *abstract* μέθοδο draw()
  - Τώρα η Shape είναι abstract, και δε μπορεί να παράγει αντικείμενα
  - Η μεταβλητή figure δε μπορεί να περιέχει αντικείμενο της Shape γιατί είναι αδύνατο να δημιουργηθεί τέτοιο αντικείμενο
  - Οποιοδήποτε αντικείμενο (όπως ένα αντικείμενο της κλάσης Star) που ανήκει σε υποκλάση της Shape θα διαθέτει μια μέθοδο draw()
  - Ο compiler της Java μπορεί να είναι σίγουρος ότι η κλήση figure.draw() θα είναι σε κάθε περίπτωση νόμιμη και δεν παράγει syntax error

Ένα πρόβλημα

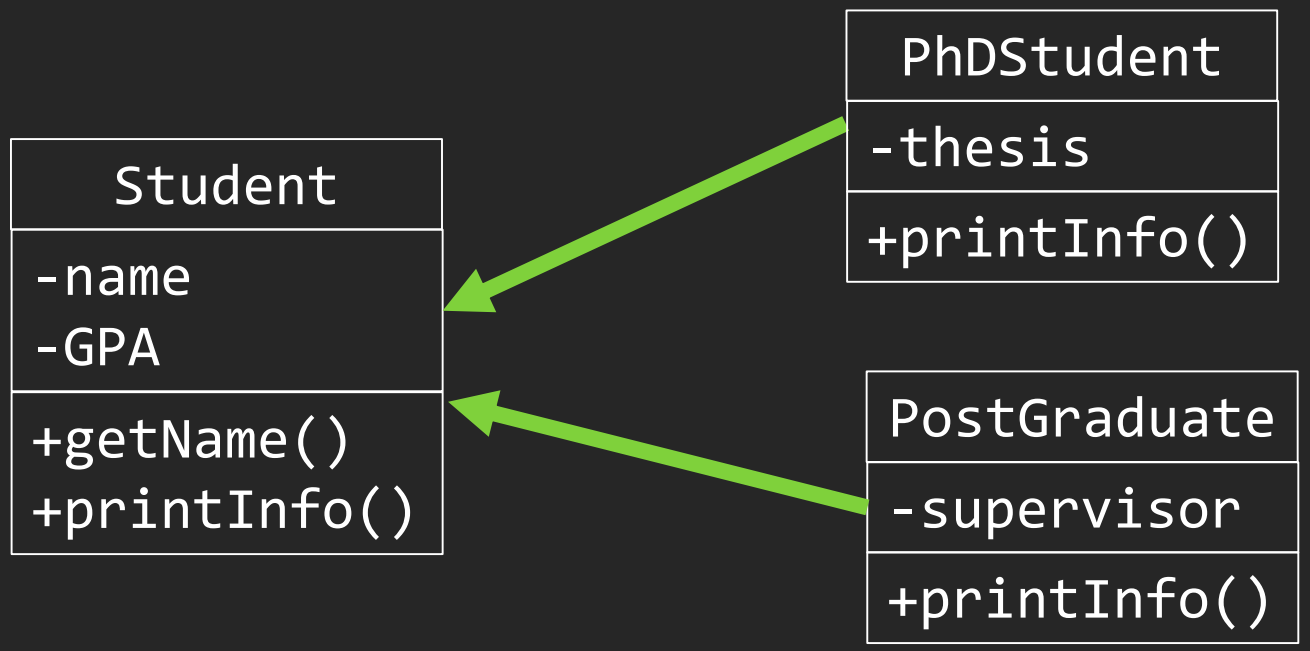
- class Shape { … }
- class Star extends Shape {
  void draw() { … }
  …
  }
- class Circle extends Shape {
  void draw() { … }
  …
  }
- Shape someShape = new Star();
  - Νόμιμο αφού ένα αντικείμενο Star είναι και αντικείμενο Shape
- someShape.draw();
  - Προκαλείται syntax error, γιατί ένα αντικείμενο Shape μπορεί να μην έχει μέθοδο draw()
  - Remember: ***A class knows its superclass, but not its subclasses***

Μια λύση

- abstract class Shape {
  abstract void draw();
  }
- class Star extends Shape {
  void draw() { … }
  …
  }
- class Circle extends Shape {
  void draw() { … }
  …
  }
- Shape someShape = new Star();
  - Νόμιμο επειδή ένα αντικείμενο Star είναι και αντικείμενο Shape
  - Όμως τώρα το Shape someShape = new Shape(); δεν επιτρέπεται
- someShape.draw();
  - Νόμιμο, γιατί κάθε στιγμιότυπο που μπορεί να παραχθεί θα έχει υποχρωτικά μια μέθοδο draw()

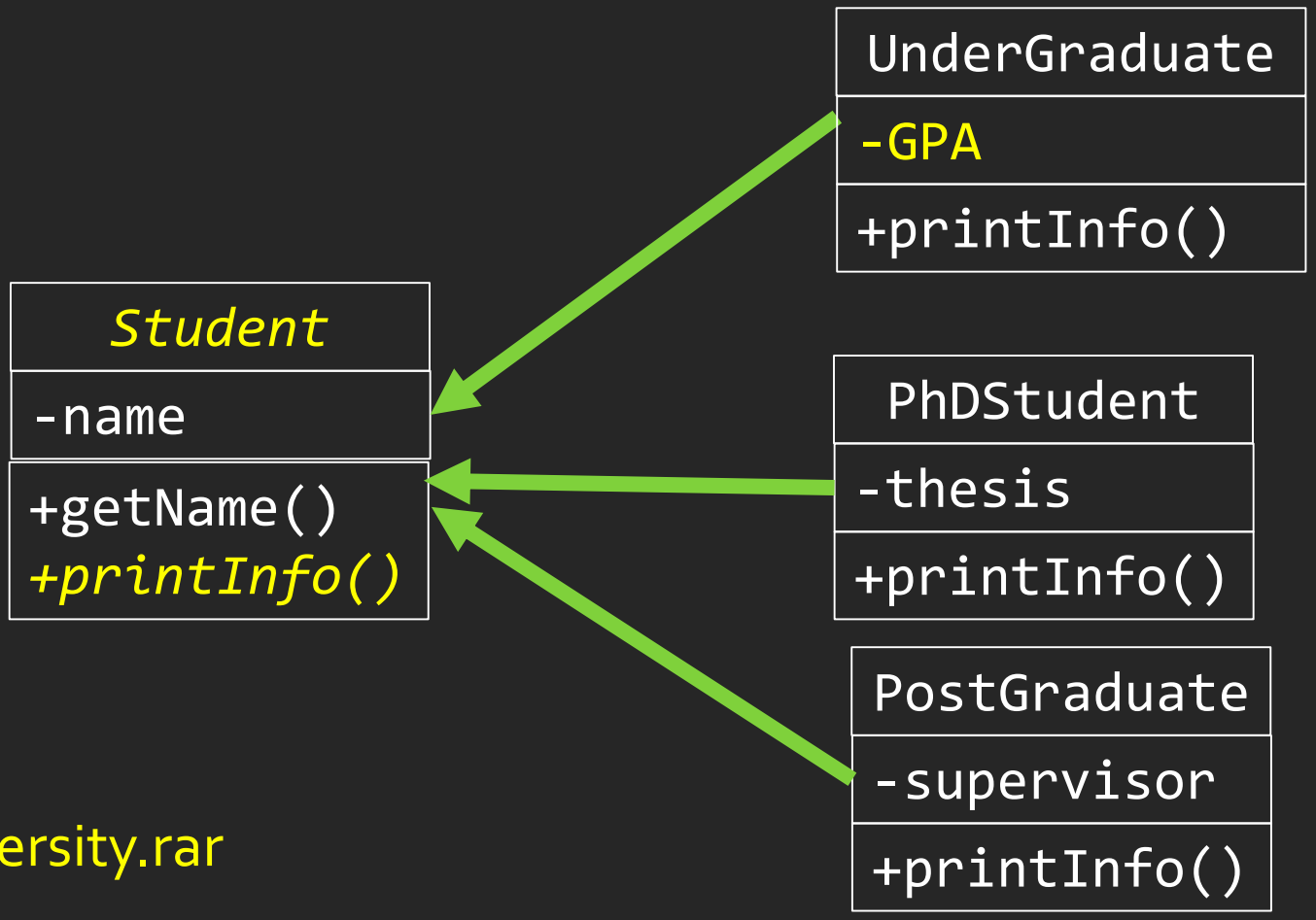
Ας δούμε ένα παράδειγμα

- Έστω ότι θέλουμε να μοντελοποιήσουμε σε ένα σύστημα τις οντότητες προπτυχιακός φοιτητής (Student), μεταπτυχιακός (PostGraduate) και διδακτορικός φοιτητής (PhDStudent)



- Αν πρέπει να αλλάξουμε τη σχεδίαση ώστε μόνο ο προπτυχιακός φοιτητής να έχει GPA?

Χρήση abstract κλάσης



University.rar

Θα μπορούσαμε να κάνουμε την printInfo() μη αφηρημένη και να την καλούμε στις υποκλάσεις?

Interfaces (Διασυνδέσεις)

- Ένα interface δηλώνει (προδιαγράφει) μεθόδους αλλά δεν παρέχει το σώμα τους (δεν τις υλοποιεί)

```
interface KeyListener {
    public void keyPressed(KeyEvent e);
    public void keyReleased(KeyEvent e);
    public void keyTyped(KeyEvent e);
}
```

- Όλες οι μέθοδοι είναι έμμεσα public και abstract
  - Μπορούμε και να το δηλώνουμε ρητά αλλά δεν υπάρχει λόγος!
- Δε μπορούμε να δημιουργήσουμε αντικείμενα από ένα interface
  - Το interface είναι σαν μια *πολύ* abstract κλάση — δεν υλοποιεί καμία από τις μεθόδους του
- Μπορεί να περιέχει και σταθερές (final μεταβλητές)

Σχεδίαση interfaces

- Συνήθως χρησιμοποιούμε έτοιμα interfaces Java που μας παρέχει η Oracle
- Σε κάποιες περιπτώσεις μπορεί να χρειαστεί να σχεδιάσουμε δικά μας interfaces
  - Π.χ. όταν θέλουμε κλάσεις διαφορετικού είδους να έχουν κάποιες κοινές δυνατότητες
- Αν θέλουμε να μπορούμε να δημιουργούμε animated απεικονίσεις όλων των αντικειμένων μια κλάσης θα ορίζαμε ένα interface:
  - ```
    public interface Animatable {
        install(Panel p);
        display();
    }
    ```
- Έτσι μπορούμε να γράψουμε κώδικα που εμφανίζει οποιοδήποτε αντικείμενο κλάσης Animatable σε ένα Panel της επιλογής μας, απλά καλώντας αυτές τις μεθόδους
    ```

Υλοποίηση ενός interface I

- Μπορούμε να κάνουμε extend μια class, αλλά να υλοποιήσουμε (να κάνουμε implement) ένα interface
- Μια κλάση μπορεί να κάνει extend μόνο μία άλλη κλάση (δηλ. να είναι υποκλάση μιας μόνο κλάσης), αλλά μπορεί να υλοποιεί όσα interfaces θέλουμε
- Παράδειγμα:

```
class MyListener
        implements KeyListener, ActionListener { …
}
```

Υλοποίηση ενός interface II

- Όταν λέμε ότι μια κλάση υλοποιεί (implements) ένα interface, πρακτικά δίνουμε την υπόσχεση ότι θα *ορίσουμε* (θα υλοποιήσουμε) μέσα στην κλάση όλες τις μεθόδους που δηλώνονται στο interface
- Π.χ.:

```
class MyKeyListener implements KeyListener {
    public void keyPressed(KeyEvent e) {...};
    public void keyReleased(KeyEvent e) {...};
    public void keyTyped(KeyEvent e) {...};
}
```

  - Στη θέση των "..." πρέπει να μπει ο κώδικας που υλοποιεί τις μεθόδους
- Στη συνέχεια μπορούμε να δημιουργήσουμε ένα αντικείμενο MyKeyListener

Μερική υλοποίηση ενός Interface

- Μπορούμε να ορίσουμε μόνο κάποιες από τις μεθόδους ενός interface αλλά όχι όλες:

```
abstract class MyKeyListener implements KeyListener {
    public void keyTyped(KeyEvent e) {...};
}
```

- Αν μια κλάση δεν παρέχει τις υλοποιήσεις όλων των μεθόδων που υποσχέθηκε είναι μια αφηρημένη κλάση
- Άρα πρέπει να τη δηλώσουμε ως abstract
- Μπορούμε να επεκτείνουμε (*extend)* ένα interface (και έτσι να του προσθέσουμε μεθόδους):
  - interface FunkyKeyListener extends KeyListener { ... }

Γιατί χρησιμοποιούμε interfaces;

- Λόγος 1: Μια κλάση μπορεί να κληρονομεί μόνο 1 άλλη κλάση αλλά μπορεί να υλοποιεί πολλά interfaces
  - Έτσι η κλάση μπορεί να παίζει διάφορους ρόλους (να υλοποιεί πολλές συμπεριφορές)
  - Ειδικά όταν υλοποιούμε Applets, είναι συνήθης πρακτική μια κλάση να υλοποιεί πολλούς διαφορετικούς listeners
  - Παράδειγμα:

```
class MyApplet extends Applet
        implements ActionListener, KeyListener {
...
}
```

- Λόγος 2: Μπορούμε να γράψουμε μεθόδους που χρησιμοποιούνται σε περισσότερες από μία κλάσεις

Πώς χρησιμοποιούμε τα interfaces

```
interface RuleSet { boolean isLegal(Move m, Board b);
                    void makeMove(Move m); }
```

Κάθε κλάση που υλοποιεί το Interface RuleSet πρέπει να διαθέτει αυτές τις δύο μεθόδους

```
class CheckersRules implements RuleSet { // Μία υλοποίηση
   public boolean isLegal(Move m, Board b) { ... }
   public void makeMove(Move m) { ... }
}
class ChessRules implements RuleSet { ... } // άλλη implementation
class LinesOfActionRules implements RuleSet { ... } // και άλλη μία
RuleSet rulesOfThisGame = new ChessRules();
```

Η εντολή ανάθεσης είναι νόμιμη γιατί ένα αντικείμενο rulesOfThisGame είναι αντικείμενο RuleSet

```
if (rulesOfThisGame.isLegal(m, b)) { makeMove(m); }
```

Νόμιμη εντολή γιατί ανεξάρτητα από το τι είδος αντικείμενο είναι το RuleSet θα διαθέτει τις μεθόδους isLegal και makeMove

Ακόμα ένα παράδειγμα

- Υλοποίηση και χρήση της κλάσης DataSet που λειτουργεί ως δομή υπολογισμού στατιστικών για τα στοιχεία που προσθέτω σε αυτή (count, sum, max, min)
  - Χωρίς χρήση αφαιρέσεων (Dataset_noAbstractions.zip)
  - Με χρήση αφαιρέσεων (Dataset_withAbstractions.zip)

instanceof

- instanceof είναι ένα keyword (τελεστής) που απαντά στο αν μια μεταβλητή είναι μέλος μια κλάσης ή ενός Interface
  - Επιστρέφει true / false
- Παράδειγμα, αν δηλώσουμε ότι

```
class Dog extends Animal implements Pet {...}
Animal fido = new Dog();
```

Τότε τα παρακάτω είναι όλα true:

```
fido instanceof Dog
fido instanceof Animal
fido instanceof Pet
```

- Το instanceof σπάνια χρησιμοποιείται
  - Κάθε φορά που χρειάζεται να χρησιμοποιήσουμε το instanceof, θα πρέπει να σκεφτούμε εάν η μέθοδος που γράφουμε θα πρέπει να μετακινηθεί στις συγκεκριμένες υποκλάσεις

Χειρισμός Εξαιρέσεων

EXCEPTIONS IN JAVA

Τι είναι μια εξαίρεση

- Έστω το κάτωθι τμήμα κώδικα:

```
int age = Integer.parseInt(input);
```

  - Προφανώς αναμένουμε έναν αριθμό (ακέραιο) από το χρήστη που αντιπροσωπεύει μια έγκυρη τιμή ηλικίας
- Θεωρήστε ωστόσο τις παρακάτω πιθανότητες:
  1. Αν ο χρήστης πληκτρολογήσει ένα "$" αντί του 4?
  2. Αν ο χρήστης εισάγει ένα δεκαδικό ψηφίο ?
  3. Αν κρατήσει πατημένο το πλήκτρο "3" για πολύ και εισαχθεί ένας πολύ μεγάλος αριθμός ?
- Δεν αναμένουμε συνθήκες όπως αυτές – Ωστόσο συμβαίνουν !!

Τι είναι μια εξαίρεση

- Ορισμένα πράγματα μπορεί να εξελιχθούν με λάθος τρόπο κατά τη διάρκεια της εκτέλεσης και δεν μπορούν να ανιχνευθούν κατά τη διάρκεια της μεταγλώττισης
- Ένα άλλο παράδειγμα: απόπειρα διαίρεσης με το 0
- Από την πλευρά του compiler δεν υπάρχει κανένα πρόβλημα με τις αντίστοιχες εντολές και τα προβλήματα θα προκύψουν μόνο κατά την εκτέλεση του προγράμματος
- Στο σημείο αυτό «ενεργοποιείται ένας συναγερμός» και η Java προσπαθεί να «παράγει μια εξαίρεση» υποδηλώνοντας ότι κάτι αντικανονικό έχει συμβεί.

Παράδειγμα

```
import java.util.*;

public class StackDemo {
  public static void main(String args[]) {
  // creating stack
  Stack st = new Stack();

 // populating stack
  st.push("Java");
  st.push("Source");
// removing top object
  System.out.println("Removed object is: "+st.pop());

  // elements after remove
  System.out.println("Elements after remove: "+st);
  }
}
```

Παράδειγμα

```
import java.util.*;

public class StackDemo {
  public static void main(String args[]) {
  // creating stack
  Stack st = new Stack();

 // populating stack
  st.push("Java");
  st.push("Source");
// removing top object
  System.out.println("Removed object is: "+st.pop());
  st.pop();
  st.pop();
  // elements after remove
  System.out.println("Elements after remove: "+st);
  }
```

Ορολογία: Actors και Actions

- **Operation - Λειτουργία**
  Μια μέθοδος που μπορεί να παράγει (raise) μια εξαίρεση.
- **Invoker**
  Μια μέθοδος που καλεί λειτουργίες και χειρίζεται τις εξαιρέσεις που προκύπτουν.
- **Exception**
  Μια ακριβής και πλήρης περιγραφή ενός αντικανονικού γεγονότος. Πρόκειται για αντικείμενα στη Java.
- **Raise (Παραγωγή Εξαίρεσης)**
  Αποστολή μιας εξαίρεσης από την operation στον invoker. (Καλείται throw στην Java). Ένας άλλος συνηθισμένος όρος είναι το emit.
- **Handle - Χειρισμός**
  Η απόκριση του Invoker στην εξαίρεση, καλείται catch στη Java.
- **Backtrack**
  Η δυνατότητα επαναφοράς των πλαισίων στοίβας στο σημείο όπου παρήχθη η εξαίρεση στο πρώτο κατάλληλο χειριστή

Ορισμένοι τύποι εξαιρέσεων

Arithmetic Exception	πρόβλημα κατά την αποτίμηση αριθμητικής παράστασης, όπως η διαίρεση με το μηδέν
NullPointerException	κλήση μεθόδου δια μέσου αναφοράς που έχει τιμή null
IndexOutOfBoundsException	ένας δείκτης πίνακα έχει βρεθεί εκτός των ορίων της δομής
EOFException	εντοπίστηκε σύμβολο τέλους αρχείου

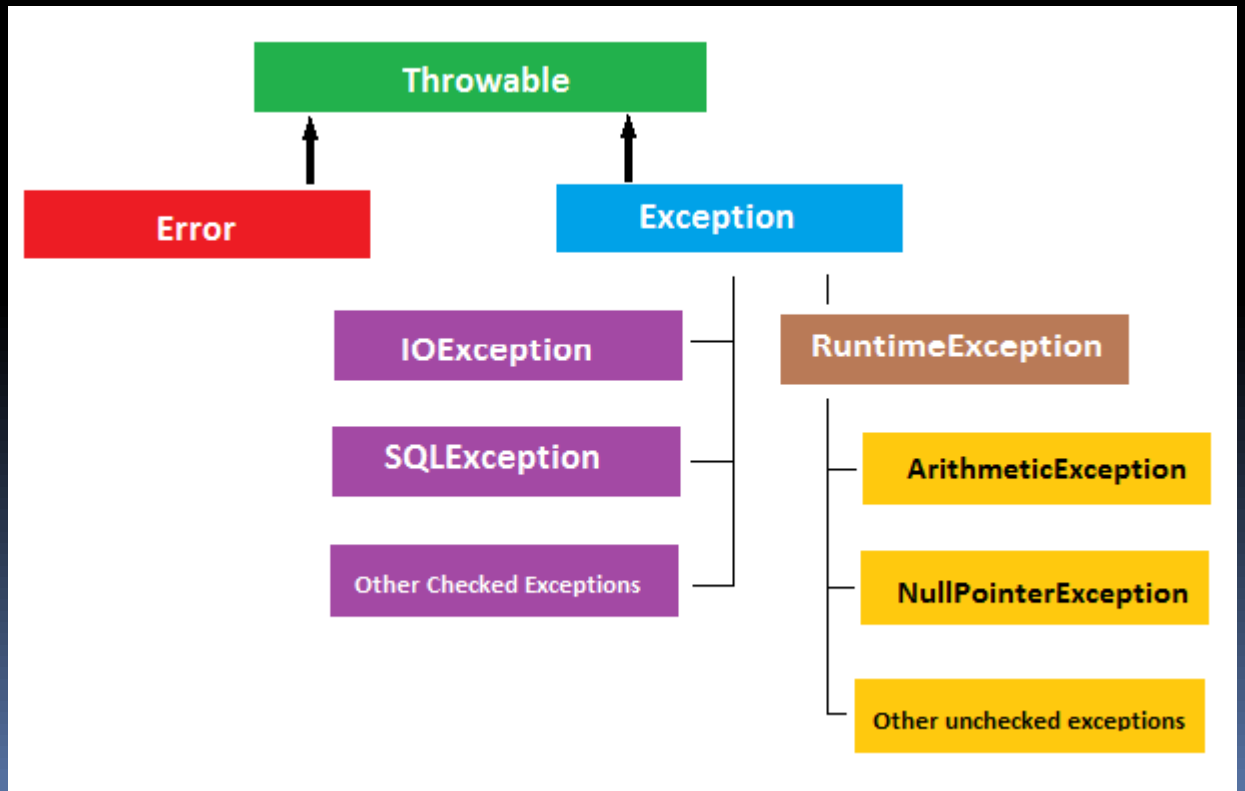
Εξαιρέσεις

- Ένα πρόγραμμα μπορεί να αντιμετωπίσει μια εξαίρεση με τρεις τρόπους
  - να την αγνοήσει
  - να την χειριστεί στο σημείο που παράγεται
  - να την χειριστεί σε κάποιο άλλο σημείο του προγράμματος

Κατηγοριοποίηση Εξαιρέσεων

- **Checked Exceptions**
  Πρέπει είτε να «συλληφθούν» (caught) από μια μέθοδο ή να δηλωθούν στην υπογραφή της

- **Unchecked Exceptions**
  Δεν είναι υποχρεωτικό να χειριστούμε αυτούς τους τύπους εξαιρέσεων.
  - *Runtime exceptions* μπορεί να προέρχονται από μεθόδους ή και από την ίδια τη JVM.
    Τα Errors παράγονται από τη JVM, και συνήθως αφορούν μια fatal state.



Κατηγοριοποίηση Εξαιρέσεων

- Unchecked exceptions
  - ArithmenticException
  - ArrayIndexOutOfBoundsExcept ion
  - ArrayStoreException
  - ClassCastException
  - IllegalMonitoStateException
  - IllegalStateException
  - IllegalThreadStateException
- Checked exceptions
  - ClassNotFoundException
  - CloneNotSupportedException
  - IllegalAccessException
  - InstantiationException
  - IterruptedException
  - NoSuchFileException
  - NoSuchMEthodException

Keywords for Java Exceptions

- **throws**
  Περιγράφει τις εξαιρέσεις που μπορεί να παραχθούν από μια μέθοδο.
- **throw**
  Παραγωγή μιας εξαίρεσης και προώθηση στον πρώτο διαθέσιμο χειριστή στην στοίβα κλήσεων.
- **try**
  Υποδηλώνει την αρχή ενός block που σχετίζεται με ένα σύνολο χειριστών εξαιρέσεων (μπορεί να παράγει εξαιρέσεις).
- **catch**
  Αν το block try παράγει μια εξαίρεση του αντίστοιχου τύπου, η ροή του ελέγχου μεταφέρεται εδώ.
- **finally**
  Καλείται πάντοτε όταν τερματίζεται το τμήμα try και μετά από οποιοδήποτε τυχόν χειρισμό στο τμήμα catch.

Try – catch – (finally)

TRY: ορίζει το μπλοκ κώδικα που μπορεί να προκαλέσει exception. Το μπλοκ αυτό ονομάζεται guarded region.

- Κάθε μπλοκ try πρέπει να περιέχει κάτι από τα παρακάτω:
  - Τουλάχιστον ένα catch block αλλά μπορεί να έχει και περισσότερα
  - Μόνο ένα finally block
  - catch blocks και ένα finally block
- Τα catch blocks προηγούνται του finally block

FINALLY: περιέχει κώδικα που εκτελείται πάντα είτε προκλήθηκε εξαίρεση, είτε όχι. Π.χ. κλείσιμο αρχείων, συνδέσεων με ΒΔ, network sockets, κτλ.

- Μόνο 1 finally block ανά εντολή try

Γενική σύνταξη

```
public void setProperty(String p_strValue) throws
   NullPointerException {
   if (p_strValue == null) { throw new NullPointerException("..."); }
}

public void myMethod(String text) {
   MyClass oClass = new MyClass();

   try {
       oClass.setProperty(text);
       oClass.doSomeWork();

   } catch (NullPointerException npe) {
   System.err.println("Unable to set property: "+npe.toString());
   } finally {
       oClass.cleanup();      }
}
```

Παράδειγμα

```
public void foo() {
    try { /* αρχή ενός block try-catch */
        int a[] = new int[2];
        a[4] = 1; /* προκαλεί runtime exception λόγω του
        index */
    } catch (ArrayIndexOutOfBoundsException e) {
        System.out.println("exception: " + e.getMessage());
        e.printStackTrace();
    }
}

/* ο κώδικας που ακολουθεί περνάει το compile, αλλά παράγει
   exception στο χρόνο εκτέλεσης. Πρόκειται για περίπτωση
   λιγότερο προφανή αλλά πολύ συνηθισμένη (an off-by-one-
   error) */
public int[] bar() {
    int a[] = new int[2];
    for (int x = 0; x <= 2; x++) { a[x] = 0; }
    return a;}
```

Πλεονεκτήματα των Exceptions

Από “The Java Tutorials”
http://docs.oracle.com/javase/tutorial/essential/exceptions/advantages.html

Πλεονέκτημα 1: Διαχωρισμός κώδικα χειρισμού σφαλμάτων από «κανονικό» κώδικα (αποφυγή “spaghetti” code)

Πλεονεκτήματα των Exceptions

Έστω το παρακάτω τμήμα (ψευδο)κώδικα

```
readFile {
    open the file;
    determine its size;
    allocate that much memory;
    read the file into memory;
    close the file;
}
```

Ο κώδικας φαίνεται αρκετά «απλός» και καθαρός αλλά αγνοεί τα παρακάτω πιθανά σφάλματα:

-Τι θα συμβεί αν το αρχείο δεν μπορεί να ανοίξει?

-Τι θα συμβεί αν το μήκος του αρχείου δεν μπορεί να προσδιοριστεί?

-Τι θα συμβεί εάν δεν μπορεί να δεσμευθεί η απαραίτητη μνήμη?

-Τι θα συμβεί εάν η ανάγνωση του αρχείου αποτύχει?

-Τι θα συμβεί εάν το αρχείο δεν μπορεί να κλείσει?

Πλεονεκτήματα των Exceptions

Θα μπορούσαμε να γράψουμε κώδικα για τον χειρισμό και την αναφορά όλων των πιθανών προβλημάτων ως εξής:

```
errorCodeType readFile {
    initialize errorCode = 0;

    open the file;
    if (theFileIsOpen) {
        determine the length of the file;
        if (gotTheFileLength) {
            allocate that much memory;
            if (gotEnoughMemory) {
                read the file into memory;
                if (readFailed) {
                    errorCode = -1;}
            } else {
                errorCode = -2;}
        } else {
            errorCode = -3;
        }
        close the file;
        if (theFileDidntClose && errorCode == 0) {
            errorCode = -4;        }
       else {
            errorCode = errorCode and -4;        }
    } else {
        errorCode = -5;    }
    return errorCode;}
```

Υπάρχει τόσος κώδικας για τον χειρισμό των σφαλμάτων που οι αρχικές 5 γραμμές κώδικα έχουν «χαθεί»

Πλεονεκτήματα των Exceptions

Ο μηχανισμός των εξαιρέσεων επιτρέπει να διατηρήσουμε ανέπαφη τη λογική του κυρίως κώδικα και να χειριστούμε τα σφάλματα σε άλλο σημείο:

```
readFile {
    try {

        open the file;
        determine its size;
        allocate that much memory;
        read the file into memory;
        close the file;

    } catch (fileOpenFailed) {
       doSomething;
    } catch (sizeDeterminationFailed) {
        doSomething;
    } catch (memoryAllocationFailed) {
        doSomething;
    } catch (readFailed) {
        doSomething;
    } catch (fileCloseFailed) {
        doSomething;
    }
}
```

Πλεονεκτήματα των Exceptions

Πλεονέκτημα 2: «Προώθηση» των σφαλμάτων προς τα πάνω στην στοίβα κλήσεων

Έστω ότι η readFile είναι η 4η μέθοδος στη σειρά σε μια ακολουθία κλήσεων μεθόδων από το κυρίως πρόγραμμα (method1->method2->method3->readFile

Έστω ότι η method1 είναι η μόνη μέθοδος που ενδιαφέρεται για τα σφάλματα που μπορεί να συμβούν στην readFile

Η συμβατική αντιμετώπιση θα ήταν

Πλεονεκτήματα των Exceptions

```
method1 {
    errorCodeType error;
    error = call method2;
    if (error)
        doErrorProcessing;
    else
        proceed;
}
```

```
errorCodeType method2 {
    errorCodeType error;
    error = call method3;
    if (error)
        return error;
    else
        proceed;
}
```

```
errorCodeType method3 {
    errorCodeType error;
    error = call readFile;
    if (error)
        return error;
    else
        proceed;
}
```

Πλεονεκτήματα των Exceptions

Αντιμετώπιση με τον μηχανισμό των exceptions

```
method1 {
    try {
        call method2;
    } catch (exception e) {
        doErrorProcessing;
    }
}
```

```
method2 throws exception {
    call method3;
}

method3 throws exception {
    call readFile;
}

readFile throws exception {
    . . . ;
}
```

Πλεονεκτήματα των Exceptions

Πλεονέκτημα 3: Κατηγοριοποίηση και Διαφοροποίηση βάσει τύπου των σφαλμάτων

Όλες οι εξαιρέσεις είναι αντικείμενα κλάσεων

Μια μέθοδος μπορεί να γράψει κώδικα για τον χειρισμό μιας συγκεκριμένης κατηγορίας σφαλμάτων

```
catch (FileNotFoundException e)
{
    ...
}
```

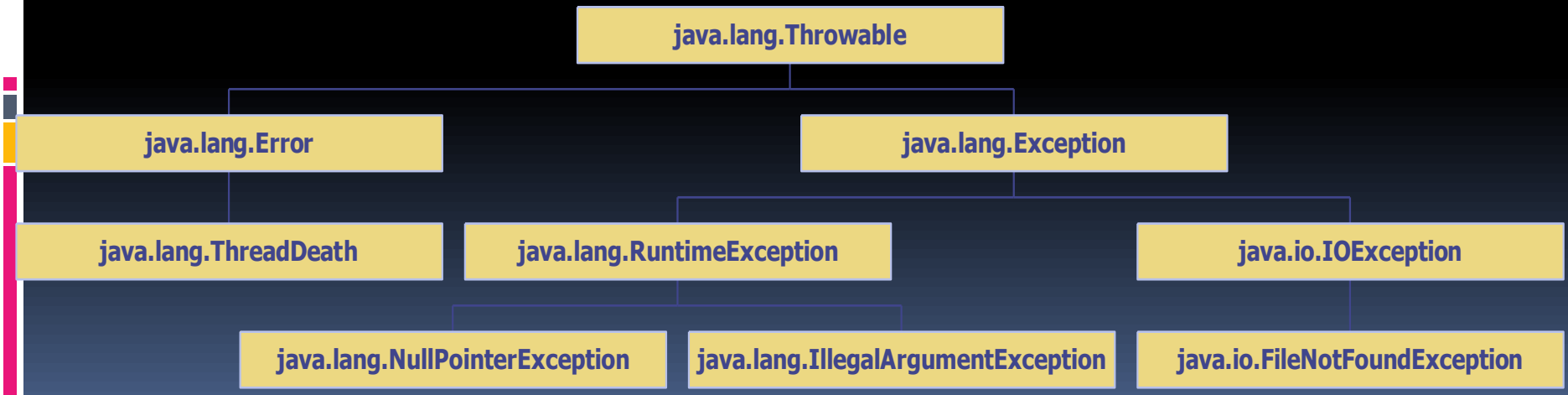
Χειρισμός μόνο σφαλμάτων που σχετίζονται με αδυναμία εύρεσης του αρχείου

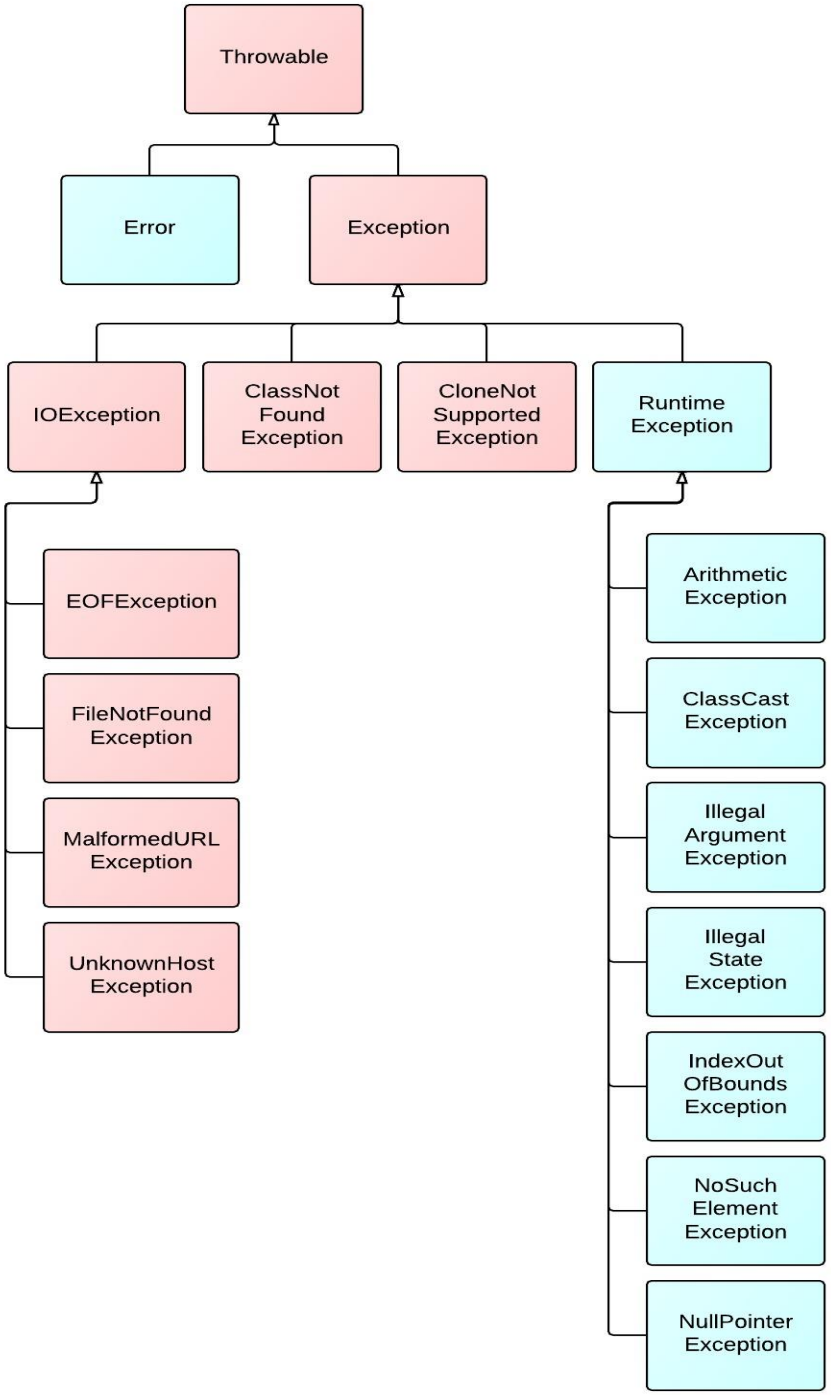
```
catch (IOException e) {
    ...
}
```

Χειρισμός όλων των I/O σφαλμάτων ανεξαρτήτως του ειδικού τους τύπου

Ιεραρχία εξαιρέσεων στη Java

Error	Ασυνήθιστες καταστάσεις που δεν προκαλούνται από λάθη του κώδικα αλλά γενικότερης φύσης προβλήματα που μπορεί να συμβούν (π.χ. JVM running out of memory). Δε μπορούμε να επανακάμψουμε με κάποιο τρόπο από ένα Error και δεν χρειάζεται να γράψουμε κώδικα για να το διαχειριστούμε. Πρακτικά τα Errors δεν είναι εξαιρέσεις (δεν προέρχονται από την κλάση Exception)
Exception	Δεν αφορά κάτι που είναι αποτέλεσμα προγραμματιστικού λάθους αλλά το ότι δεν είναι διαθέσιμος κάποιος πόρος ή κάποια άλλη συνθήκη που είναι απαραίτητη για την ορθή εκτέλεση του κώδικα. Π.χ. αν ο κώδικας πρέπει να συνδεθεί με κάποια άλλη εφαρμογή ή άλλο υπολογιστή που δεν ανταποκρίνεται.




Throwable
Error
Exception
IOException
ClassNot Found Exception
CloneNot Supported Exception
Runtime Exception
EOFException
FileNotFound Exception
MalformedURL Exception
UnknownHost Exception
Arithmetic Exception
ClassCast Exception
Illegal Argument Exception
Illegal State Exception
IndexOut OfBounds Exception
NoSuch Element Exception
NullPointer Exception

Δημιουργία δικής σας exception class

```
/* You should extend RuntimeException to create an unchecked
   exception,
 * or Exception to create a checked exception. */
class MyException extends Exception {

  /* This is the common constructor. It takes a text
   argument. */
  public MyException(String p_strMessage) {
    super(p_strMessage);
  }

  /* A default constructor is also a good idea! */
  public MyException () {
    super();
  }
```

Κληρονομικότητα και Exceptions

- Όταν παράγεται μια εξαίρεση σε ένα try block η java αντιμετωπίζει τα πολλαπλά catch statements σαν τα cases σε μια switch εντολή
  - Το πρώτο catch που ικανοποιεί το είδος της εξαίρεσης θα εκτελεστεί και τα υπόλοιπα τα προσπερνάει
  - Είναι σημαντικό να τοποθετούμε τα subclass exceptions ΠΡΙΝ από τα superclass exceptions αλλιώς προκαλείται συντακτικό λάθος

```
import java.io.IOException;

class TesteExceptions
{
  public static void main (String args[])
  {
    int num1 = 100;
    int num2 = 50;
    int num3 = 50;
    int result1;

    try
    {
        result1 = num1/(num2-num3);
        System.out.println("Result1 = " +result1);
    }
    catch (Exception e)
    { System.out.println("This is a mistake");}
    catch (ArithmeticException g)
    {System.out.println("Dinision by zero");}
   }
  }
```

Έξοδος:

Syntax error: Arithmetic Exception has already been caught

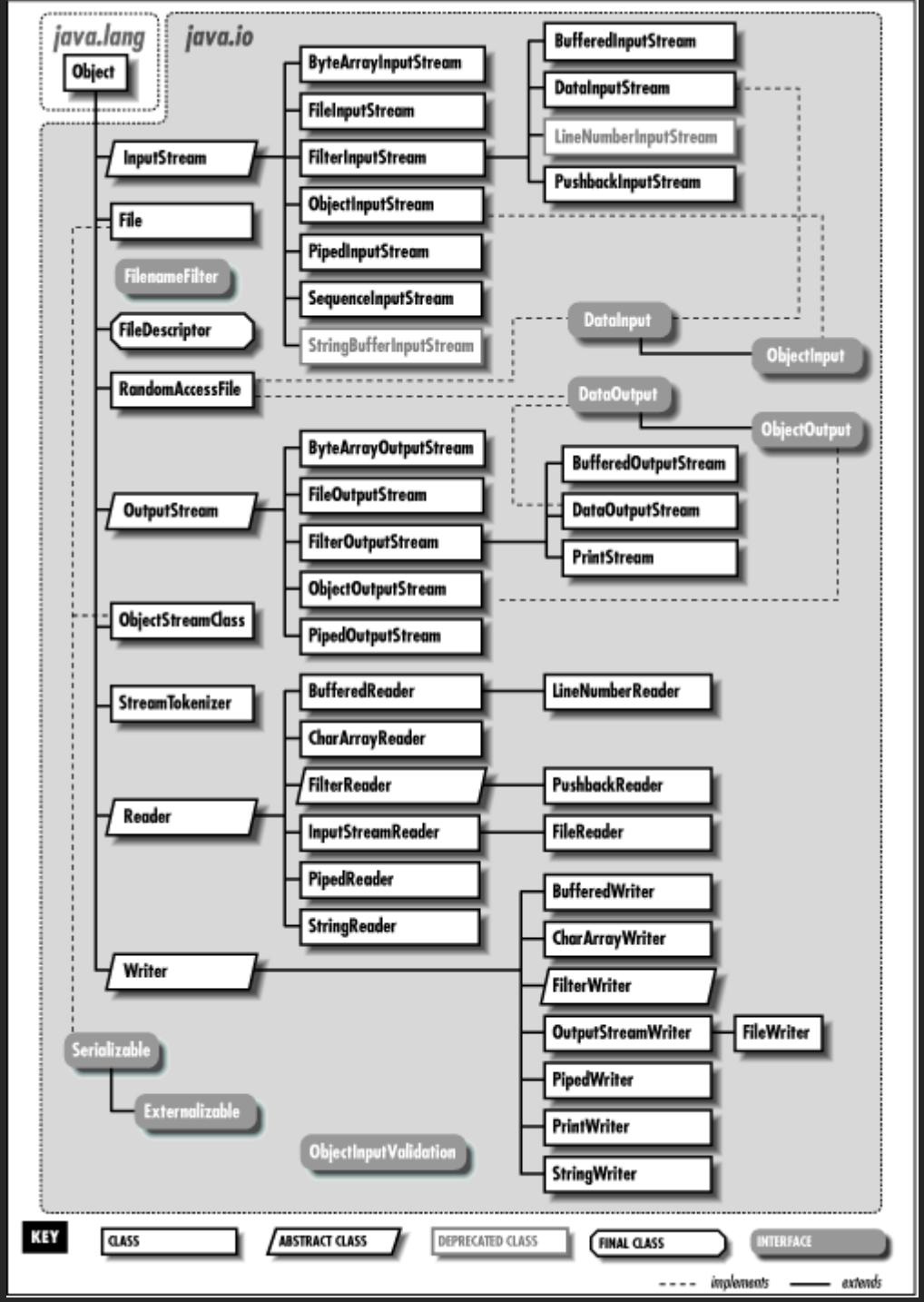
Οντοκεντρικός Προγραμματισμός

ΦΡΟΝΤΙΣΤΗΡΙΟ JAVA

Αρχεία δεδομένων

- Το πακέτο java.io περιλαμβάνει περισσότερες από 60 κλάσεις και διασυνδέσεις για το χειρισμό αρχείων δεδομένων.

- Αρκετές από τις κλάσεις του πακέτου java.io εντάσσονται σε μία από τις δύο κύριες κατηγορίες, οι οποίες διαχειρίζονται
  - αρχεία κειμένου
  - δυαδικά αρχεία


java.lang
java.io
Object
InputStream
File
FilenameFilter
FileDescriptor
RandomAccessFile
OutputStream
ObjectStreamClass
StreamTokenizer
Reader
Writer
Serializable
Externalizable
ObjectInputValidation
ByteArrayInputStream
FileInputStream
FilterInputStream
ObjectInputStream
PipedInputStream
SequenceInputStream
StringBufferInputStream
BufferedInputStream
DataInputStream
LineNumberInputStream
PushbackInputStream
DataInput
ObjectInput
DataOutput
ObjectOutput
ByteArrayOutputStream
FileOutputStream
FilterOutputStream
ObjectOutputStream
PipedOutputStream
BufferedOutputStream
DataOutputStream
PrintStream
BufferedReader
CharArrayReader
FilterReader
InputStreamReader
PipedReader
StringReader
LineNumberReader
PushbackReader
FileReader
BufferedWriter
CharArrayWriter
FilterWriter
OutputStreamWriter
FileWriter
PipedWriter
PrintWriter
StringWriter
KEY
CLASS
ABSTRACT CLASS
DEPRECATED CLASS
FINAL CLASS
INTERFACE
implements
extends

Αρχεία κειμένου

- Όταν μεταφέρουμε δεδομένα από ένα πρόγραμμα σε ένα αρχείο κειμένου, τότε όλα τα δεδομένα μετατρέπονται σε κείμενο πριν από την αποθήκευση.
  - Στην ουσία, τα αρχεία κειμένου περιλαμβάνουν δεδομένα σε μορφή παρόμοια με τον τύπο char της Java — συνήθως απλές, οργανωμένες σε γραμμές, αλφαριθμητικές πληροφορίες τις οποίες μπορεί να διαβάσει ο χρήστης χρησιμοποιώντας ένα συντάκτη κειμένου.
- Τα αρχεία κειμένου μπορούν να διαβαστούν και από προγράμματα που έχουν γραφεί σε άλλες γλώσσες προγραμματισμού, εκτός από την Java.

Δυαδικά αρχεία

- Όταν μεταφέρουμε δεδομένα από ένα πρόγραμμα σε ένα δυαδικό αρχείο, τότε τα δεδομένα δεν υφίστανται καμία μετατροπή πριν από την αποθήκευση.
- Τα δυαδικά αρχεία ποικίλουν περισσότερο: συνηθισμένο παράδειγμα είναι τα αρχεία εικόνων, και τα εκτελέσιμα προγράμματα.
- Σε ένα δυαδικό αρχείο μπορούμε να αποθηκεύσουμε με μία λειτουργία εγγραφής ακόμα και ένα ολόκληρο αντικείμενο ή συλλογή αντικειμένων, διαδικασία γνωστή ως σειριοποίηση.
- Ένα δυαδικό αρχείο μπορεί να διαβαστεί μόνο από προγράμματα όπου τα δεδομένα αναπαρίστανται εσωτερικά με τον ίδιο τρόπο όπως στο πρόγραμμα που δημιούργησε το συγκεκριμένο αρχείο.

Αρχεία δεδομένων

- Οι κλάσεις που αναφέρονται στη διαχείριση αρχείων κειμένου ονομάζονται
  - αναγνώστες (readers)
  - και γραφείς (writers)
- Οι κλάσεις που αναφέρονται στη διαχείριση δυαδικών αρχείων είναι γνωστές ως χειριστές ρευμάτων (stream).
- Σε αρκετές περιπτώσεις, ωστόσο, χρησιμοποιείται ο όρος ρεύμα δεδομένων είτε αναφερόμαστε σε είσοδο/έξοδο από/σε δυαδικά αρχεία είτε σε αρχεία κειμένου. Στη δεύτερη περίπτωση αναφερόμαστε σε ρεύματα κειμένου.

Αντικείμενα αρχείων – η κλάση File

- Τα αντικείμενα αρχείων χρησιμοποιούνται για να πάρουμε πληροφορίες για φακέλους ή αρχεία.
- Τα αντικείμενα αρχείων δεν χρησιμοποιούνται για την ανάγνωση ή εγγραφή σε φακέλους ή αρχεία. Γι' αυτό το λόγο χρησιμοποιούνται τα ρεύματα εισόδου/εξόδου.
- Για τη δημιουργία αντικειμένων αρχείων χρησιμοποιούμε την κλάση File.

Αντικείμενα αρχείων – η κλάση File

Μέθοδος	Περιγραφή
`exists()`	Επιστρέφει **`true`** αν υπάρχει το αντικείμενο αρχείου ή φακέλου, και **`false`** διαφορετικά
`isDirectory()`	Επιστρέφει **`true`** αν το αντικείμενο αναφέρεται σε φάκελο, και **`false`** διαφορετικά
`isFile()`	Επιστρέφει **`true`** αν το αντικείμενο αναφέρεται σε αρχείο, και **`false`** διαφορετικά
`isAbsolute()`	Επιστρέφει **`true`** αν το αντικείμενο αναφέρεται σε μια πλήρη διαδρομή, και **`false`** διαφορετικά
`canRead()`	Επιστρέφει **`true`** αν μπορούμε να διαβάσουμε από το αρχείο, και **`false`** διαφορετικά
`canWrite()`	Επιστρέφει **`true`** αν μπορούμε να γράψουμε στο αρχείο, και **`false`** διαφορετικά
`getName()`	Επιστρέφει ένα αλφαριθμητικό, το οποίο αντιπροσωπεύει το όνομα του αρχείου ή το όνομα του φακέλου ανάλογα, χωρίς να αναφέρεται η διαδρομή
`getPath()`	Επιστρέφει ένα αλφαριθμητικό με το όνομα της διαδρομής
`length()`	Επιστρέφει έναν ακέραιο τύπου long που εκφράζει το μήκος του αρχείου σε **`bytes`**

Αντικείμενα αρχείων – η κλάση File

Για τη δημιουργία ενός αντικειμένου αρχείου:

- Δημιουργούμε πρώτα ένα αντικείμενο που αντιστοιχεί στη διαδρομή του αρχείου (path):

```
File myDir = new File("C:\\projects\\files");
```

Το "\\" είναι ο χαρακτήρας διαφυγής "\" μαζί με το διαχωριστικό διαδρομής "\"

- Στη συνέχεια, δημιουργούμε ένα αντικείμενο αρχείου που αντιστοιχεί στο ίδιο το αρχείο data.txt:

```
File myFile = new File(myDir, "data.txt");
```

- Μπορούμε να συνδυάσουμε τις δύο εργασίες ως εξής:

```
File myFile = new File("C:\\projects\\files", "data.txt");
```

Ρεύματα δεδομένων

- Σε ένα πρόγραμμα που διαχειρίζεται αρχεία δεδομένων, μπορούμε να φανταστούμε τα δεδομένα που ανταλλάσσονται μεταξύ του προγράμματος και ενός αρχείου ως ένα ρεύμα.
- Τα δεδομένα ρέουν από μια προέλευση (source) προς ένα προορισμό (destination).
- Αρχικά, «ανοίγουμε» το ρεύμα.
- Αν το ρεύμα ρέει από το πρόγραμμα προς το αρχείο, τότε «γράφουμε» στο ρεύμα.
- Αν το ρεύμα ρέει από αρχείο προς το πρόγραμμα, τότε το πρόγραμμα «διαβάζει» δεδομένα.
- Τέλος, «κλείνουμε» το ρεύμα.

Ρεύματα δεδομένων

Χρησιμοποιώντας την καθιερωμένη ορολογία, διακρίνουμε τα εξής βήματα:

- Άνοιγμα αρχείου: αντιστοιχίζουμε ένα αντικείμενο ρεύματος στο αρχείο από το οποίο θέλουμε να διαβάσουμε ή στο οποίο θέλουμε να γράψουμε.
- Εγγραφή/ανάγνωση: πραγματοποιείται η επικοινωνία με το φυσικό αρχείο δεδομένων διαβιβάζοντας μηνύματα εγγραφής και ανάγνωσης στο αντικείμενο ρεύματος που έχουμε δημιουργήσει. Στην ουσία, το φυσικό αρχείο δεδομένων «κρύβεται» μέσα στο αντικείμενο ρεύματος.
- Κλείσιμο αρχείου: κλείσιμο του αντικειμένου ρεύματος και τερματισμός της σύνδεσης με το φυσικό αρχείο δεδομένων. Ταυτόχρονα, απελευθερώνονται οι εσωτερικοί πόροι που είχαν κατανεμηθεί στο αντικείμενο ρεύματος. Αν ξεχάσουμε να κλείσουμε ένα ρεύμα στο οποίο γράφει το πρόγραμμα, τότε υπάρχει κίνδυνος απώλειας δεδομένων.

Ρεύματα δεδομένων (3)

- Όπως ήδη αναφέρθηκε, υπάρχουν πολλές κλάσεις στο πακέτο java.io για τη δημιουργία αντικειμένων ρεύματος.
- Η δημιουργία ενός αντικειμένου ρεύματος γίνεται σε αρκετές περιπτώσεις με έναν ιδιαίτερο τρόπο: ένα στιγμιότυπο μιας κλάσης ρεύματος μπορεί να χρησιμοποιηθεί ως όρισμα στον κατασκευαστή μιας άλλης κλάσης ρεύματος. Η διαδικασία αυτή μπορεί να επαναληφθεί αρκετές φορές μέχρι να καταλήξουμε σε ένα ρεύμα που θα μας ικανοποιεί.

Έξοδος κειμένου: η κλάση FileWriter (1)

- Η αποθήκευση δεδομένων σε αρχείο γίνεται, όπως ήδη αναφέρθηκε, σε τρία βήματα:

  1. Άνοιγμα του αρχείου

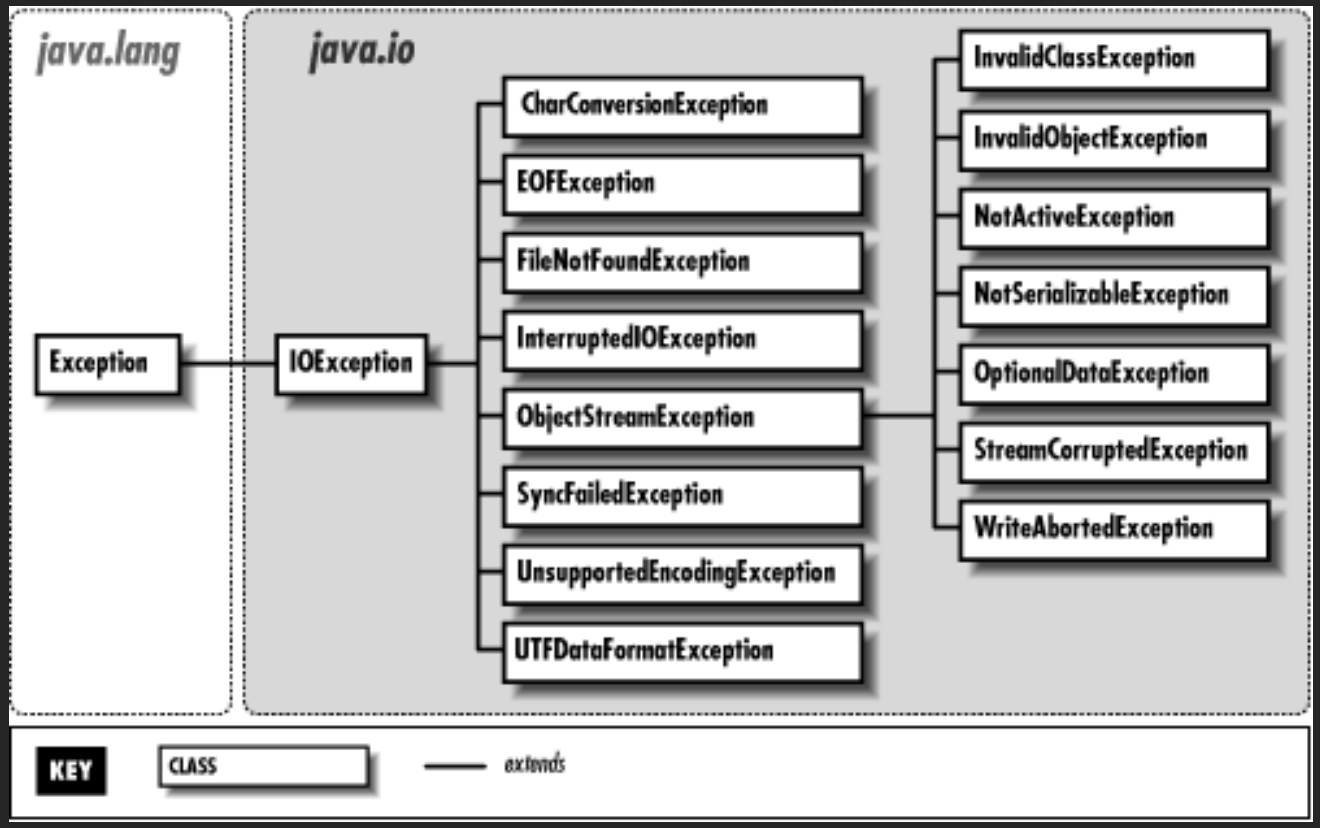
  2. Εγγραφή των δεδομένων

  3. Κλείσιμο του αρχείου

- Οποιοδήποτε από αυτά τα βήματα μπορεί να αποτύχει για διαφορετικούς λόγους, πολλοί από τους οποίους είναι πέρα από τον έλεγχο του προγραμματιστή της εφαρμογής. Για παράδειγμα, μπορεί να υπάρχει αδυναμία ανοίγματος του αρχείου ή εγγραφής στο σκληρό δίσκο κτλ.

- Συνεπώς, πρέπει να αναμένουμε την παραγωγή εξαιρέσεων σε οποιοδήποτε από τα παραπάνω στάδια.

Ιεραρχία κλάσεων εξαιρέσεων



Έξοδος κειμένου: η κλάση FileWriter (2)

Βήμα 1ο: άνοιγμα αρχείου

- Δημιουργούμε ένα αντικείμενο **FileWriter**, του οποίου ο κατασκευαστής δέχεται το όνομα του αρχείου.

```
FileWriter writer = new FileWriter("…όνομα αρχείου…");
```

- Αν το αρχείο δεν υπάρχει ήδη, δημιουργείται.
- Αν το αρχείο υπάρχει και θέλουμε να γίνει εγγραφή μετά από όσα υπάρχουν ήδη στο αρχείο, τότε χρησιμοποιούμε τον παρακάτω κατασκευαστή

```
public FileWriter(String filename, boolean append)
```

με δεύτερο όρισμα την τιμή true.

- Το όνομα του αρχείου μπορεί να έχει τη μορφή αλφαριθμητικού ή ενός αντικειμένου **File**:

```
File f = new File("…όνομα αρχείου…");
```

Έξοδος κειμένου: η κλάση FileWriter (3)

Βήμα 1ο: άνοιγμα αρχείου

- η δημιουργία ενός αντικειμένου **`FileWriter`** έχει ως αποτέλεσμα το άνοιγμα του εξωτερικού αρχείου και την προετοιμασία του για τη λήψη της εξόδου.
- Αν το άνοιγμα του αρχείου αποτύχει για οποιονδήποτε λόγο, τότε ο κατασκευαστής θα παραγάγει μια εξαίρεση **`IOException`**.

Έξοδος κειμένου: η κλάση FileWriter (3)

Βήμα 2ο: εγγραφή δεδομένων

- Όταν ένα αρχείο ανοίξει επιτυχώς, τότε μπορεί να χρησιμοποιηθεί η μέθοδος **`write`** του εγγραφέα για την αποθήκευση χαρακτήρων — συχνά με μορφή αλφαριθμητικών — στο αρχείο:

```
writer.write("…τμήμα κειμένου…");
```

- Ακόμα και αν το αρχείο ανοίξει επιτυχώς, υπάρχει πιθανότητα να αποτύχει οποιαδήποτε προσπάθεια εγγραφής σε αυτό.

Έξοδος κειμένου: η κλάση FileWriter (4)

Βήμα 3ο: κλείσιμο αρχείου

- Όταν τελειώσει η εγγραφή της εξόδου, πρέπει να κλείσουμε κανονικά το αρχείο:

```
writer.close();
```

- Αυτό εξασφαλίζει ότι όλα τα δεδομένα έχουν πράγματι γραφεί στο εξωτερικό σύστημα αρχείων, και έχει συχνά ως αποτέλεσμα την αποδέσμευση κάποιων εσωτερικών ή εξωτερικών πόρων.
- Πάλι, υπάρχει πιθανότητα (αν και μικρή) να αποτύχει η προσπάθεια να κλείσουμε το αρχείο.

Έξοδος κειμένου: η κλάση FileWriter (5)

```
try
{
   File f = new File("... όνομα αρχείου ...");
   FileWriter writer = new FileWriter(f);
   while(υπάρχει επιπλέον κείμενο για εγγραφή)
   {
      ...
      writer.write(επόμενο τμήμα κειμένου);
      ...
   }
   writer.close();
}
catch(IOException e)
{
   προέκυψε κάποιο λάθος στην προσπέλαση του αρχείου
}
```

Είσοδος κειμένου: η κλάση FileReader (1)

- Το διάβασμα δεδομένων από αρχείο γίνεται, όπως ήδη αναφέρθηκε, σε τρία βήματα:

1. Άνοιγμα του αρχείου
2. Ανάγνωση των δεδομένων
3. Κλείσιμο του αρχείου

Είσοδος κειμένου: η κλάση FileReader (2)

- Ενώ οι φυσικές μονάδες για την εγγραφή κειμένου είναι οι χαρακτήρες και τα αλφαριθμητικά, οι αντίστοιχες για την ανάγνωση του κειμένου είναι οι **χαρακτήρες και οι γραμμές**.
- Ωστόσο, παρόλο που η κλάση **`FileReader`** περιλαμβάνει μια μέθοδο για την ανάγνωση ενός χαρακτήρα, δεν περιλαμβάνει κάποια μέθοδο για την ανάγνωση μίας γραμμής.
- Το πρόβλημα με την ανάγνωση γραμμών από αρχείο είναι ότι δεν υπάρχει κάποιο προκαθορισμένο όριο για το μήκος γραμμής.
- Αυτό σημαίνει ότι οποιαδήποτε μέθοδος χρησιμοποιηθεί για την επιστροφή της επόμενης ολόκληρης γραμμής από ένα αρχείο πρέπει να μπορεί να διαβάσει έναν οσοδήποτε μεγάλο αριθμό χαρακτήρων.

Είσοδος κειμένου: η κλάση FileReader (3)

- Γι' αυτό, συνήθως, το αντικείμενο **FileReader** περιτυλίγεται σε ένα αντικείμενο **BufferedReader**:

```
FileReader freader = new FileReader("…όνομα αρχείου…");
BufferedReader reader = new BufferedReader(freader);
```

ή συνοπτικά

```
BufferedReader reader = new BufferedReader(
                    new FileReader("…όνομα αρχείου…"));
```

- Ένα αντικείμενο **BufferedReader** διαθέτει τη μέθοδο **readLine** για το διάβασμα μιας γραμμής.

- Ο χαρακτήρας τερματισμού γραμμής αφαιρείται πάντα από το αλφαριθμητικό που επιστρέφεται
- Το τέλος ενός αρχείου υποδηλώνεται με μια τιμή **null**.

Είσοδος κειμένου: η κλάση FileReader (4)

- Η κλάση **BufferedReader**, εκτός από το γεγονός ότι διαθέτει τη μέθοδο **readLine**, παρουσιάζει και ένα άλλο σημαντικό πλεονέκτημα:
- Η κλάση **BufferedReader** χρησιμοποιεί μια **περιοχή προσωρινής αποθήκευσης** (buffer), δηλαδή μια περιοχή στη μνήμη που λειτουργεί ως προσωρινός αποθηκευτικός χώρος.
- Το τυπικό μέγεθος μιας περιοχής προσωρινής αποθήκευσης που χρησιμοποιείται για την ενδιάμεση αποθήκευση δεδομένων από ένα αρχείο σε ένα πρόγραμμα Java είναι 8192 χαρακτήρες.
- Με τη χρήση του buffer η προσπέλαση του δίσκου είναι λιγότερο συχνή, κάτι που αυξάνει την ταχύτητα του προγράμματος.

Είσοδος κειμένου: η κλάση FileReader (5)

```
try
{
  File f = new File("... όνομα αρχείου ...");
  BufferedReader reader = new BufferedReader(new FileReader(f));
  String line = reader.readLine();
  while(line != null)
  {
      κάποια ενέργεια με τη γραμμή
      line = reader.readLine();
  }
  reader.close();
}
catch(FileNotFoundException e)
{
   δεν εντοπίστηκε το συγκεκριμένο αρχείο
}
catch(IOException e)
{
   κάποιο πρόβλημα προέκυψε στην ανάγνωση ή στο κλείσιμο
}
```

Παράδειγμα εγγραφής σε αρχείο κειμένου

- fileWriter
- fileReader

Σειριοποίηση αντικειμένων (1)

- Η **σειριοποίηση** επιτρέπει την εγγραφή και την ανάγνωση ενός ολόκληρου αντικειμένου σε και από ένα εξωτερικό αρχείο αντίστοιχα με μία λειτουργία.
- Η σειριοποίηση επιτρέπει επίσης να γραφούν και να διαβαστούν αντικείμενα όχι μόνο σε ένα σύστημα αρχείων, αλλά και σε ένα δίκτυο.
- Η σειριοποίηση λειτουργεί τόσο με απλά αντικείμενα όσο και με αντικείμενα πολλών στοιχείων, όπως οι συλλογές.
- Με την σειριοποίηση αποφεύγεται η ανάγνωση και η εγγραφή αντικειμένων πεδίο προς πεδίο.

Σειριοποίηση αντικειμένων (2)

- Προκειμένου να είναι κατάλληλη για σειριοποίηση, μια κλάση πρέπει να υλοποιεί τη διασύνδεση **`Serializable`** η οποία ορίζεται στο πακέτο **`java.io`**.
- Η διασύνδεση **`Serializable`** δεν ορίζει μεθόδους (marking interface).
- Αυτό σημαίνει ότι γίνεται αυτόματη διαχείριση της διαδικασίας σειριοποίησης από το σύστημα χρόνου εκτέλεσης, και από το χρήστη απαιτείται απλά να γράψει λίγες γραμμές κώδικα.

Έξοδος σε δυαδικό αρχείο (1)

- Η αποθήκευση αντικειμένων σε αρχείο γίνεται σε τρία βήματα:

  1. Άνοιγμα του αρχείου
  2. Εγγραφή των δεδομένων
  3. Κλείσιμο του αρχείου

- Κάθε αντικείμενο που εμπλέκεται πρέπει να προέρχεται από μια κλάση που υλοποιεί τη διασύνδεση Serializable.

- Για να γράψουμε αντικείμενα σε ένα αρχείο πρέπει να δημιουργήσουμε ένα αντικείμενο ObjectOutputStream, ακολουθώντας τα παρακάτω βήματα:

Έξοδος σε δυαδικό αρχείο (2)

Βήμα 1ο: άνοιγμα αρχείου για εγγραφή αντικειμένων

1. Δημιουργούμε ένα αντικείμενο αρχείου `File`:

```
File f = new File("filename.txt");
```

2. Δημιουργούμε ένα αντικείμενο **`FileOutputStream`** χρησιμοποιώντας ως όρισμα το αντικείμενο **`File`** που δημιουργήσαμε παραπάνω:

```
FileOutputStream fouts = new FileOutputStream(f);
```

3. Δημιουργούμε ένα αντικείμενο **`ObjectOutputStream`** χρησιμοποιώντας ως όρισμα το αντικείμενο `fouts`

```
ObjectOutputStream douts = new ObjectOutputStream(fouts);
```

Οι δύο τελευταίες εντολές μπορούν να συνδυαστούν σε μία:

```
ObjectOutputStream douts = new ObjectOutputStream(
                                    new FileOutputStream(f));
```

Έξοδος σε δυαδικό αρχείο (3)

Βήμα 2ο: εγγραφή αντικειμένων

- Για την εγγραφή των αντικειμένων χρησιμοποιείται η μέθοδος **`writeObject`**:

```
douts.writeObject(αντικείμενο);
```

Βήμα 3ο: κλείσιμο αρχείου

```
douts.close();
```

Παράδειγμα: Serializ1

Είσοδος από δυαδικό αρχείο (1)

- Η ανάγνωση αντικειμένων από αρχείο γίνεται σε τρία βήματα:

  1. Άνοιγμα του αρχείου

  2. Ανάγνωση των αντικειμένων

  3. Κλείσιμο του αρχείου

- Κάθε αντικείμενο που εμπλέκεται πρέπει να προέρχεται από μια κλάση που υλοποιεί τη διασύνδεση **`Serializable`**.

- Για να διαβάσουμε αντικείμενα από ένα αρχείο πρέπει να δημιουργήσουμε ένα αντικείμενο **`ObjectInputStream`**, ακολουθώντας τα παρακάτω βήματα:

Είσοδος από δυαδικό αρχείο (2)

Βήμα 1ο: άνοιγμα αρχείου για διάβασμα αντικειμένων

1. Δημιουργούμε ένα αντικείμενο αρχείου **File**:

```
File f = new File("filename.txt");
```

2. Δημιουργούμε ένα αντικείμενο **FileInputStream** χρησιμοποιώντας ως όρισμα το αντικείμενο **File** που δημιουργήσαμε παραπάνω:

```
FileInputStream fins = new FileInputStream(f);
```

3. Δημιουργούμε ένα αντικείμενο **ObjectInputStream** χρησιμοποιώντας ως όρισμα το αντικείμενο **fins**

```
ObjectInputStream dins = new ObjectInputStream(fins);
```

Οι δύο τελευταίες εντολές μπορούν να συνδυαστούν σε μία:

```
ObjectInputStream dins =

new ObjectIputStream(new FileInputStream(f));
```

Είσοδος από δυαδικό αρχείο (3)

Βήμα 2ο: διάβασμα αντικειμένων

- Για την ανάγνωση/διάβασμα των αντικειμένων χρησιμοποιείται η μέθοδος **`readObject`** της κλάσης **`ObjectInputStream`**.
- Η μέθοδος **`readObject`** επιστρέφει ένα αντικείμενο τύπου **`Object`**, οπότε χρειάζεται να γίνει casting για την προσαρμογή του επιστρεφόμενου αντικειμένου στον κατάλληλο τύπο:

```
(τύπος αντικειμένου) dins.readObject();
```

- ***Βήμα 3ο: κλείσιμο αρχείου***

```
dins.close();
```

Παράδειγμα: Serializ2