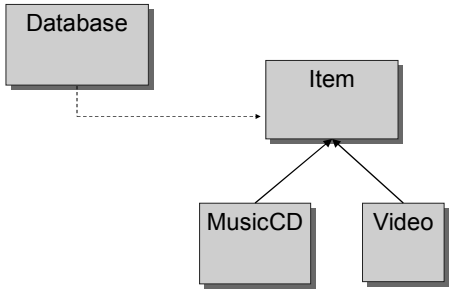
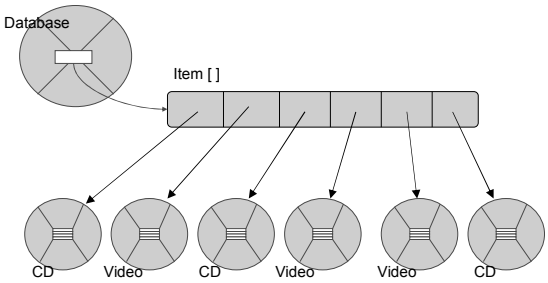
Σχεδίαση-Ανάπτυξη Εφαρμογών Πληροφορικής - Εβδομάδα 2

Εβδομάδα 2:
Υπο-τύποι και πολυμορφισμός
[sub-typing and polymorphism]

Διάγραμμα κλάσεων [Class diagram]



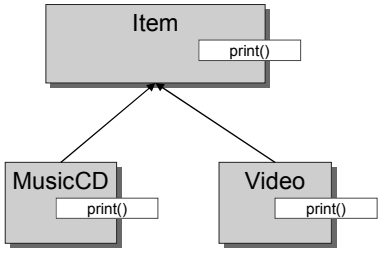
Διάγραμμα αντικειμένων [Object diagram]



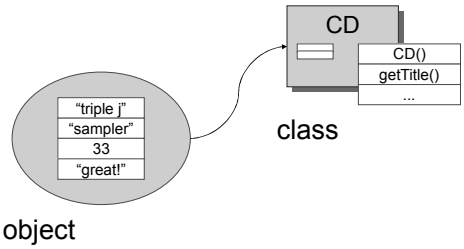
Πηγαίος κώδικας

```
void list()
{
   for (int i = 0; i < myItems.length; i++)
   {
      myItems[i].print();
   }
}
```

Σύνοψη: μέθοδος `print`



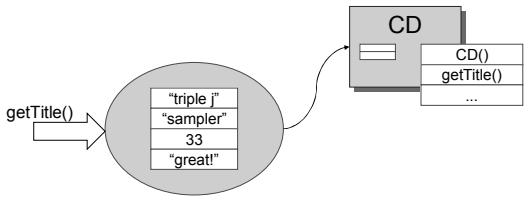
Υπενθύμιση: αντικείμενα και κλάσεις



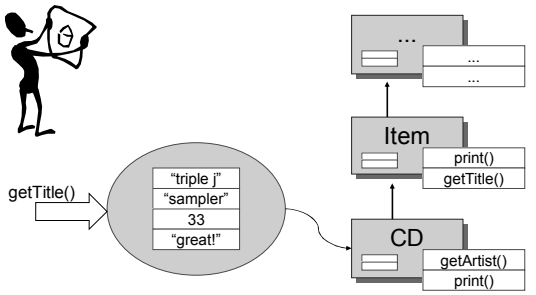
Αντώνιος Συμβώνης, ΣΕΜΦΕ, Τομέας Μαθηματικών, ΕΜΠ

Σχεδίαση-Ανάπτυξη Εφαρμογών Πληροφορικής - Εβδομάδα 2

Κλήση μεθόδου



Αντιστοίχιση μεθόδου με κληρονομικότητα



"super"

- Μία μέθοδος της υπερκλάσης μπορεί να κληθεί από μια μέθοδο της κλάσης μέσω της ειδικής μεταβλητής **`super`**.

```
public void print()
{
      super.print();
      System.out.println(...);
}
```


Χρήση της "super" σε κατασκευαστές

- Οι κατασκευαστές των υποκλάσεων πρέπει να έχουν μία κλήση προς τον "super" (τον κατασκευαστή της υπερκλάσης) στην πρώτη γραμμή του κώδικά τους.
- Εάν η κλήση έχει παραληφθεί, η εντολή
  `super();`
  εισάγεται αυτόματα από τον μεταφραστή.
- Προσοχή: Ο κατασκευαστής που δεν έχει παραμέτρους είναι αυτός που επιλέγεται να εισαχθεί αυτόματα!
- Σύσταση: Πάντα να εισάγετε μία κλήση προς τον κατασκευαστή *super* – μην εξαρτάστε από την αυτόματη εισαγωγή του

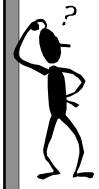
Παραδείγματα κατασκευαστών

```
MusicCD (String title)
{
      super();
      ...
}

MusicCD (String title)
{
      ...
}

MusicCD (String title)
{
      super(title);
      ...
}
```

- Άμεση κλήση στον **`super`** χωρίς παραμέτρους
- Έμμεση κλήση στον **`super`** χωρίς παραμέτρους
  (δουλεύει μόνο όταν υπάρχει στην υπερκλάση κατασκευαστής χωρίς παραμέτρους)
- Άμεση κλήση στον **`super`** με παραμέτρους

Κληρονομικότητα και υπο-τύποι

Η κληρονομικότητα δημιουργεί μία σχέση **υπο-τύπων** [subtypes] μεταξύ κλάσεων



Υπενθύμιση: Η Java είναι μία γλώσσα με **στατικό σύστημα τύπων δεδομένων** [typed language].
Μόνο τιμές με συμβατούς τύπους δεδομένων (που είναι γνωστοί κατά την μετάφραση) μπορεί να καταχωρηθούν σε μεταβλητές και παραμέτρους

Σχεδίαση-Ανάπτυξη Εφαρμογών Πληροφορικής - Εβδομάδα 2

Χρήση υπο-τύπων

```
Person aPerson;
Student aStudent;
Staff aStaffMember;

aStudent = new Student (...);
aStaffMember = new Staff (...);
```

```
aPerson = aStaffMember;
aPerson = aStudent;
aPerson.changeAddress (...);
```
Σωστή χρήση

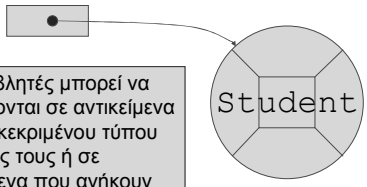
```
aStudent = aPerson;
aPerson.graduate();
```
Λάθος χρήση

Υπο-τύποι και συμφωνία τύπων [conformance]

- Αντικείμενα υπο-τύπων μπορεί να υποκαθιστούν αντικείμενα των υπερ-τύπων τους (μπορεί να χρησιμοποιηθούν σε περιπτώσεις όπου αναμένονταν ένα αντικείμενο του υπερ-τύπου).
- «Ένα αντικείμενο υπο-τύπου **είναι-ένα** [is-a] αντικείμενο υπερ-τύπου»
- Ένας υπο-τύπος **συμφωνεί** [conforms to] με τον υπερ-τύπο.

Στατικοί – δυναμικοί τύποι δεδομένων

Person aPerson;



Οι μεταβλητές μπορεί να αναφέρονται σε αντικείμενα του συγκεκριμένου τύπου δήλωσης τους ή σε αντικείμενα που ανήκουν σε οποιονδήποτε υπο-τύπο του.

Στατικοί – δυναμικοί τύποι δεδομένων

Ο **στατικός τύπος** [static type] είναι ο τύπος δήλωσης μίας μεταβλητής ή παραμέτρου όπως αυτός εμφανίζεται στον πηγαίο κώδικα.

Ο **δυναμικός τύπος** [dynamic type] είναι ο τύπος του αντικειμένου (κατά το χρόνο εκτέλεσης).

Υπο-τύποι / πολυμορφισμός

Οι υπο-τύποι παρέχουν τις ίδιες μεθόδους αλλά οι **υλοποιήσεις** [implementations] των μεθόδων μπορεί να διαφέρουν!

Δυναμικός καθορισμός τύπου [Dynamic dispatch]

Κώδικας: Αποτέλεσμα:

```
anItem = aCD;
...
anItem.print();
```

```
CD: Triple J Hottest 100 (79 min)
    artist: sampler, 33 tracks
    double CD - great!
```

Ίδιος κώδικας, Διαφορετικό αποτέλεσμα!

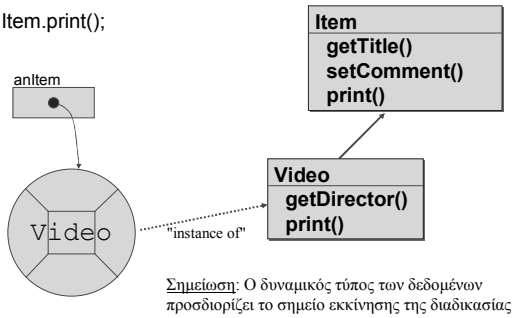
```
anItem = aVideo;
...
anItem.print();
```

```
Video: The South Park Movie (102 min)
    director: Fred Smith
    (not seen yet)
```

Σχεδίαση-Ανάπτυξη Εφαρμογών Πληροφορικής - Εβδομάδα 2

Προσδιορισμός μεθόδου [method lookup, binding]

anItem.print();



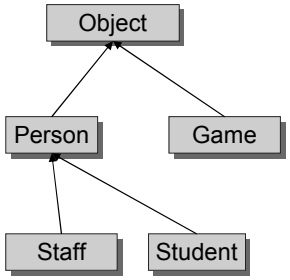
Σημείωση: Ο δυναμικός τύπος των δεδομένων προσδιορίζει το σημείο εκκίνησης της διαδικασίας προσδιορισμού της κατάλληλης μεθόδου!

Επεκτασιμότητα [Extendability]

Ο δυναμικός προσδιορισμός του τύπου δεδομένων υποστηρίζει την **επεκτασιμότητα** – νέες υποκλάσεις μπορεί να προστεθούν αργότερα χωρίς να είναι απαραίτητη η τροποποίηση του κώδικα που χρησιμοποιεί τις κλάσεις βάσης.

Η κλάση "`Object`"

- (Σχεδόν) κάθε κλάση έχει μία υπερ-κλάση.
- Εάν η υπερ-κλάση δεν έχει άμεσα δηλωθεί, τότε ως υπερκλάση θεωρείται η κλάση **`Object`**
- Οι μέθοδοι της κλάσης **`Object`** είναι διαθέσιμες σε κάθε κλάση



"`toString`"

- Η κλάση **`Object`** παρέχει τη μέθοδο **`toString`**, η οποία μετατρέπει ένα αντικείμενο σε ένα String.
- Παράδειγμα:

```
String s = person.toString();
```

- Η μέθοδος **`toString`** καλείται έμμεσα κατά την εκτέλεση της συνένωσης συμβολοσειρών (**+**)
- Παράδειγμα:

```
System.out.println("Details: " + person);
```

Συνέπεια: όλα τα αντικείμενα μπορεί να λάβουν μέρος σε συνένωση συμβολοσειρών – κατάλληλη λειτουργία για παρουσίαση αποτελεσμάτων!

Μετατροπείς πρόσβασης [Access Modifiers]

- Οι μετατροπείς πρόσβασης καθορίζουν την ορατότητα [visibility] ενός πεδίου ή μεθόδου.

Παραδείγματα:

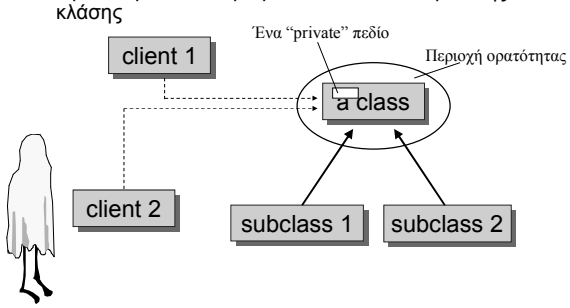
```
private int number;
protected String name;

public void changeAddress(Address newAddress)
{ ... }
private int calculateResult(int parameter)
{ ... }
```

νέο!

Μετατροπείς πρόσβασης: private, protected, public.

"`private`"

- Ορατά/προσπελάσιμα μόνο από το εσωτερικό της κλάσης

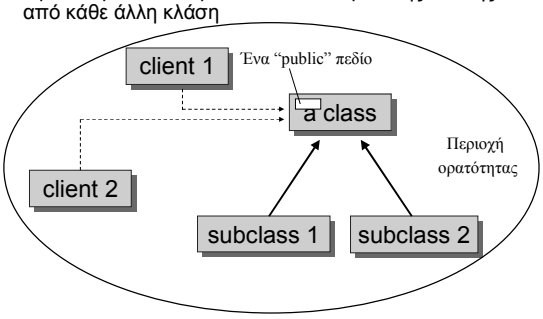


Αντώνιος Συμβώνης, ΣΕΜΦΕ, Τομέας Μαθηματικών, ΕΜΠ

Σχεδίαση-Ανάπτυξη Εφαρμογών Πληροφορικής - Εβδομάδα 2

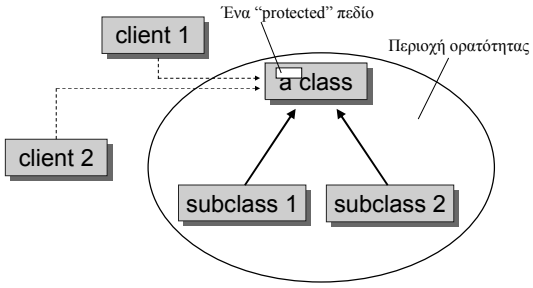
"public"

- Ορατά/προσπελάσιμα από το εσωτερικό της κλάσης και από κάθε άλλη κλάση



"protected"

- Ορατά/προσπελάσιμα από το εσωτερικό της κλάσης και από κάθε υποκλάση της



Οδηγίες χρήσης μετατροπέων πρόσβασης

- Χρησιμοποιείτε πάντοτε ένα μετατροπέα πρόσβασης.
- Περιορίστε την πρόσβαση **όσο το δυνατό** περισσότερο.
- Μη χρησιμοποιείτε **`public`** πεδία.

Η δεσμευμένη λέξη "`final`"

- Δηλώστε μία μέθοδο ως **`final`** για να αποτρέψετε τον εκ' νέου ορισμό της [prevent redefinition, overriding]
- Δηλώστε μία κλάση ως **`final`** για να είναι όλες οι μέθοδοί της "**`final`**".

Παραδείγματα:

```
final public String getPassword()
{ ... }

final class SecurityManager
{ ...
}
```


Το πρόβλημα του «αντίστροφου πολυμορφισμού»

```
class A {
  ...
}
```

```
class B extends A {
  ...
  void doSomething(int n)
  { ... }
}
```

```
// variables
A  a;
B  b;
...
b = new B();
a = b;
a.doSomething(42);

b = a;
b.doSomething(42);
```

Πως εκτελούμε μία B-μέθοδο μετά από καταχώρηση στο αντικείμενο a (τύπου A);

«λανθασμένα» στις περισσότερες γλώσσες

Ανομοιογενείς συλλογές αντικειμένων

- Μία ανομοιογενής συλλογή [heterogenous collection] είναι μία συλλογή από αντικείμενα διαφορετικών τύπων (πολυμορφική συλλογή [polymorphic collection])
- Ανομοιογενής συλλογές δημιουργούνται δηλώνοντας τα στοιχεία τους ως μέλη μιας υπερ-κλάσης τους.
- Η γενικότερη περίπτωση: Τα στοιχεία είναι τύπου "**`Object`**" – η συλλογή μπορεί να περιέχει οποιοδήποτε αντικείμενο (πχ. ListArray)

Σχεδίαση-Ανάπτυξη Εφαρμογών Πληροφορικής - Εβδομάδα 2

Απώλεια τύπου [type loss]

- Το πρόβλημα: η απώλεια τύπου

```
String note = "consider this!";
ArrayList notes = new ArrayList();
notes.addElement(note);
...
String note;
note = notes.elementAt(1);
```

Σωστό -- η παράμετρος είναι τύπου `Object`

Λάθος -- Καταχώρηση `Object` σε `String`!

Μετατροπή τύπου [casting]

- Η λύση: μετατροπή τύπου [casting]

```
String note = "consider this!";
ArrayList notes = new ArrayList();
notes.addElement(note);
...
String note;
note = (String)notes.elementAt(1);
```

Μετατροπή σε `String`!

Μία μετατροπή τύπου μετατρέπει τον στατικό τύπο ενός αντικειμένου σε έναν άλλο τύπο (συνήθως υπο-τύπο). Είναι σωστή μόνο εάν ο δυναμικός τύπος του αντικειμένου είναι σύμφωνος (conforms) με τον τύπο της μεταβλητής στην οποία καταχωρείται.

Επίδειξη

- Πολυμορφική συλλογή (πχ. ArrayList)
- Εισαγωγή στοιχείων
- Καταχώρηση με μετατροπή τύπου
- `println` χωρίς μετατροπή τύπου

  Συζήτηση: γιατί λειτουργεί σωστά;

Τμήμα κώδικα

```
String note = "consider this!";
ArrayList notes = new ArrayList();
notes.addElement(note);
...
System.out.println(notes.elementAt(1));
```

Δουλεύει; γιατί; / γιατί όχι;