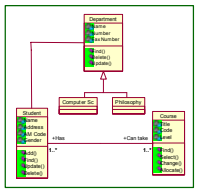
HY351:
Ανάλυση και Σχεδίαση Πληροφοριακών Συστημάτων
Information Systems Analysis and Design

Μοντελοποίηση Δομής
(Structural Modeling)



Γιάννης Τζίτζικας

Διάλεξη : 9
Ημερομηνία :
Θέμα :

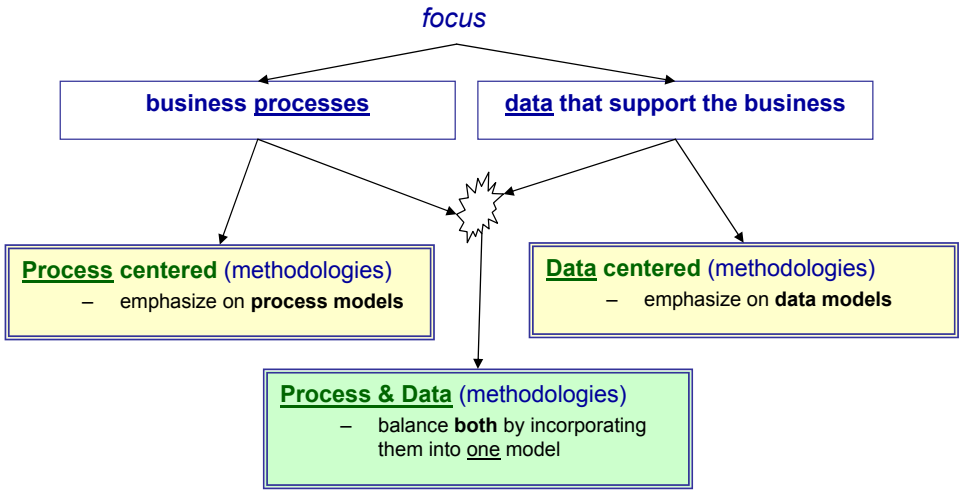
Διάρθρωση

- Μοντελοποίηση Δομής (Structural modeling)
- CRC Cards
- Εισαγωγή στα Διαγράμματα Κλάσεων (Class Diagrams)
  - Classes (Κλάσεις)
  - Attributes (Γνωρίσματα)
  - Operations (Λειτουργίες)
  - Associations (Συσχετίσεις)
  - Generalization (Γενίκευση)
  - Constraints (Περιορισμοί)
- Object Diagrams (Διαγράμματα Αντικειμένων)

Τι είναι η Μοντελοποίηση Δομής;

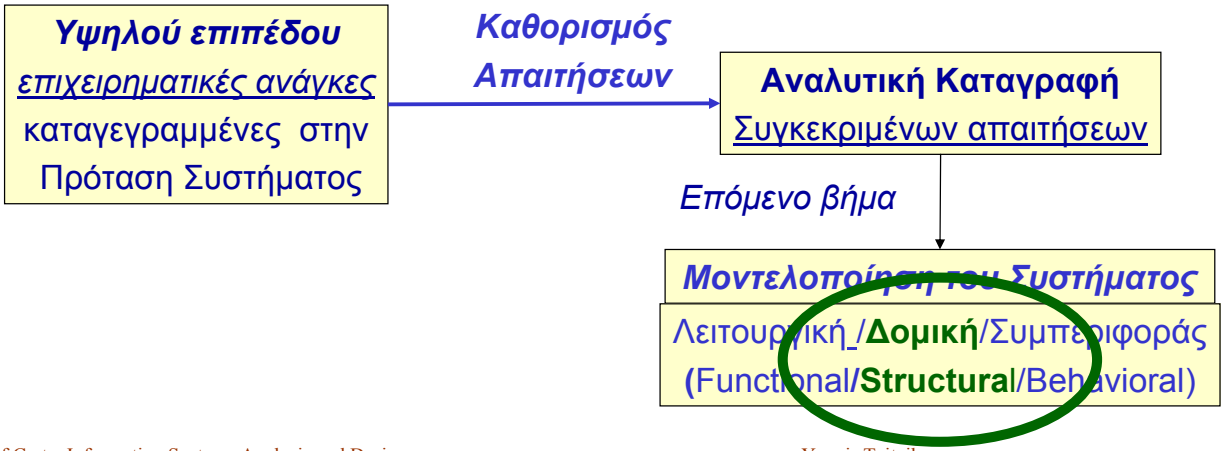
Ο σκοπός της είναι να περιγράψει:
- την δομή των δεδομένων που υποστηρίζουν τις επιχειρηματικές διαδικασίες.

Θα εστιάσουμε στην *Αντικειμενοστρεφή προσέγγιση (δεδομένα μαζί με λειτουργίες)*



Γιατί να μοντελοποιήσουμε τη δομή;

- Μειώνει το «σημασιολογικό χάσμα» μεταξύ του πραγματικού κόσμου και του κόσμου του λογισμικού
- Παριστάνει πράγματα, ιδέες και έννοιες που είναι σημαντικές για το πεδίο εφαρμογής
- Ορίζει ένα κοινό λεξιλόγιο για τους αναλυτές, σχεδιαστές και τους χρήστες



Πως μοντελοποιούμε τη δομή στην Αντικειμενοστρεφή Ανάλυση και Σχεδίαση;

Μπορούμε να χρησιμοποιούμε τους εξής τύπους μοντέλων:

- ***CRC Cards (κάρτες CRC)***
  - Συλλαμβάνουν τα ουσιώδη στοιχεία μιας κλάσης
- ***Class Diagrams (Διαγράμματα Κλάσεων)***
  - Περιγράφουν τους τύπους των αντικειμένων στο σύστημα και τις στατικές συσχετίσεις που υπάρχουν μεταξύ τους
- ***Object Diagrams (Διαγράμματα Αντικειμένων)***
  - Δείχνουν παραδειγματικούς σχηματισμούς αντικειμένων (στιγμιοτύπων, όχι κλάσεων)

Υποσημείωση: Μπορούμε να ορίσουμε διαγράμματα κλάσεων από διαφορετικές προοπτικές (several perspectives).

CRC Cards

(Class-Responsibility-Collaboration Cards)

(Κάρτες Κλάσης-Ευθύνης-Συνεργασίας)

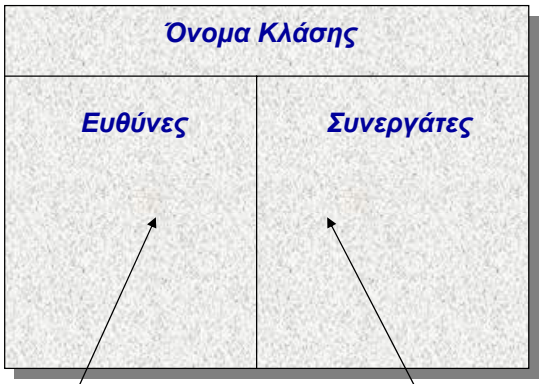
Σκοπός

- Είναι ένας άτυπος τρόπος για αντικειμενοστρεφή μοντελοποίηση
- Χρησιμοποιείται από ομάδες (σε συσκέψεις ανταλλαγής ιδεών (brain-storming))
- Προτάθηκε από το Ward Cunnigham στα τέλη της δεκαετίας του 80

Οι Κάρτες CRC μπορεί να βοηθήσουν:
- Το εντοπισμό και καθορισμό των κλάσεων
- Τον ορισμό και κατανόηση του τρόπου με τον οποίο θα συνεργάζονται

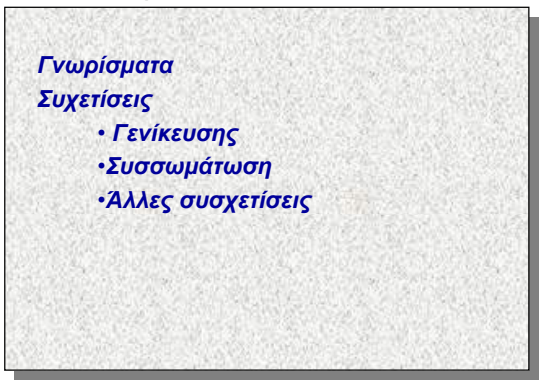
Τι είναι μια κάρτα CRC;

Μπροστινή πλευρά



Μέγεθος:10 x 15 cm

Πίσω πλευρά

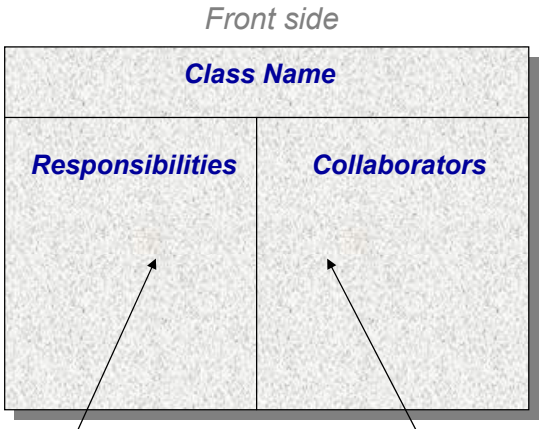


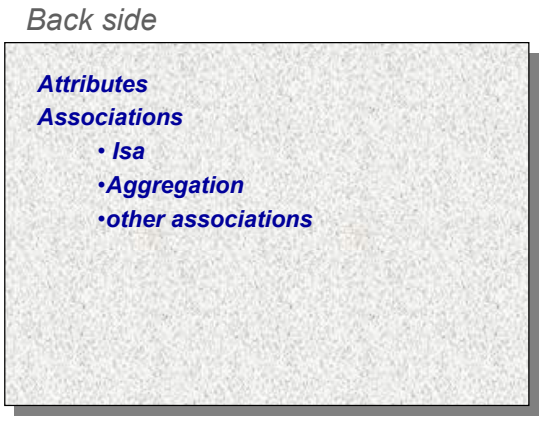
Ευθύνες:
- of ***Knowing***
- of ***Doing***

Συνεργάτες:
- Objects ***working together*** to service a request
  - i.e. UML associations

Μπορούμε να θεωρήσουμε την μπροστινή πλευρά ως την δημόσια πληροφορία, και την πίσω ως τις ενθυλακωμένες λεπτομέρειες υλοποίησης

Τι είναι μια κάρτα CRC;



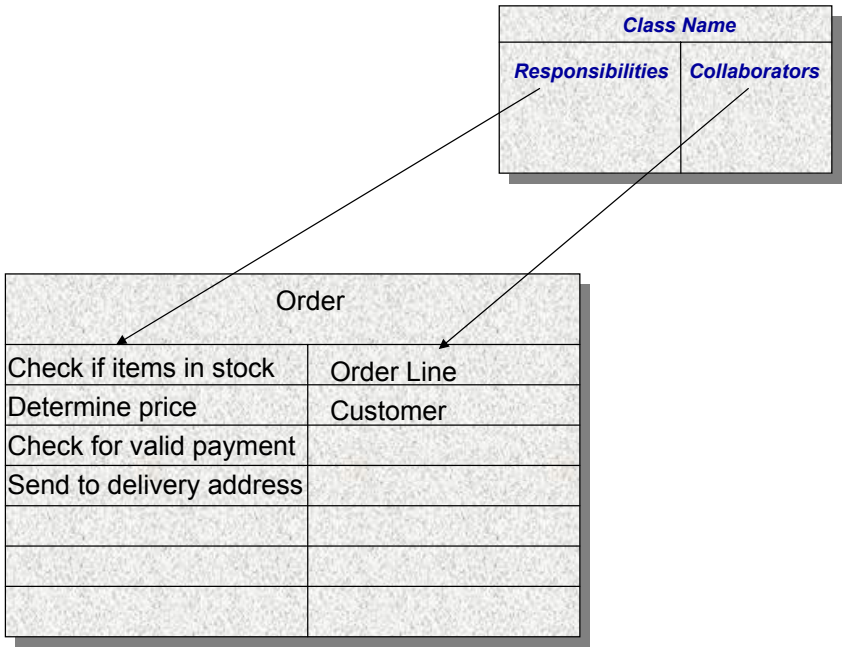


Responsibilities:
- of ***Knowing***
- of ***Doing***

Collaborators:
- Objects ***working together*** to service a request
  - i.e. UML associations

Consider the front as the public information, and the back as the encapsulated, implementation details

Παράδειγμα κάρτας CRC



Ένα παράδειγμα με 7 κάρτες

	Responsiblity	Collaborators
Teacher	Teaches Lessons Evaluates Students	Secretary Student Principal
Student	Learns Lessons	Teacher Principal
Principal	Administers Funds Diciplines Students Hires Staff	Teacher Secretary Student
Nurse	Gives First Aid Gives Vacinations	Students Teachers
Secretary	Answers Phone Prints Handouts	Teacher Principal
Janitor	Cleans Building Fixes Equipment	Teacher Secretary
Cook	Prepares Meals	Janitor

Grade school example

Taken from Cunnigham (Tektronix)

Τι μπορούμε να κάνουμε με αυτές τις κάρτες;

- Οι κάρτες αυτές παριστάνουν τη **στατική** (δομική) όψη των κλάσεων του συστήματος
- Η **δυναμική** όψη των κλάσεων μπορεί ατύπως να περιγραφεί με το λεγόμενο "***role-playing***"
  - Άλλες τεχνικές για περιγραφή της δυναμικής συμπεριφοράς dynamic behavior):
    - Διαγράμματα Ακολουθίας (Sequence Diagrams)
      - (θα μιλήσουμε για αυτά στην ενότητα Μοντελοποίηση Συμπεριφοράς)

Δουλεύοντας με τις Κάρτες
Working with CRC cards

...“Card Playing”:

- The team (<7 persons) sits around a table
  - domain experts, analysts, oo developers
- they start by identifying a number of classes of the problem domain
- they create one card for each class
  - the responsibilities should not be too many (they should fit in the card)
- they can then start role-playing the scenarios of the Use Cases
  - each person can role-play one ore more cards
  - they pick up on the air the classes that are active
  - they move them to show the exchange of messages
- If something doesn’t seem right, they change accordingly the cards (by changing their contents, or by creating/destroying cards)

Πως ξεκινάμε;

- Ένα **καλό σημείο** για ανάλυση CRC είναι οι Περιπτώσεις Χρήσης (Use Cases).

- Ξεκινήστε προσπαθώντας να εντοπίσετε τις κλάσεις του πεδίου εφαρμογής.
  - Χρησιμοποιήστε το **Έγγραφο Απαιτήσεων** και εντοπίστε τις κλάσεις που φαίνεται να ανήκουν στο πεδίο (και ανήκουν στο τμήμα του προβλήματος για το οποίο γίνεται η συνάντηση)
    - Βρείτε όλα τα **ουσιαστικά** και τα **ρήματα** στην περιγραφή του προβλήματος
    - Τα **ουσιαστικά** μπορεί να μας δείξουν ποιες είναι οι **κλάσεις**, ενώ τα **ρήματα** μπορεί να μας δείξουν ποιες είναι οι **ευθύνες** (responsibilities)

Παράδειγμα: Εντοπισμός κλάσεων αναλύοντας κείμενο (identifying the classes by analyzing the text)

Υποψήφιες κλάσεις

Problem statement.

- *This application will support the operations of a technical library for an R&D organization. This includes the searching for and lending of technical library materials, including books, videos, and technical journals. Users will enter their company ids in order to use the system; and they will enter material ID numbers when checking out and returning items.*
- *Each borrower can be lent up to five items. Each type of library item can be lent for a different period of time (books 4 weeks, journals 2 weeks, videos 1 week). If returned after their due date, the library user's organization will be charged a fine, based on the type of item(books $1/day, journals $3/day, videos $5/day).*

Παράδειγμα: Εντοπισμός Ευθυνών (Identifying the responsibilities)

Υποψήφιες ευθύνες

Problem statement.

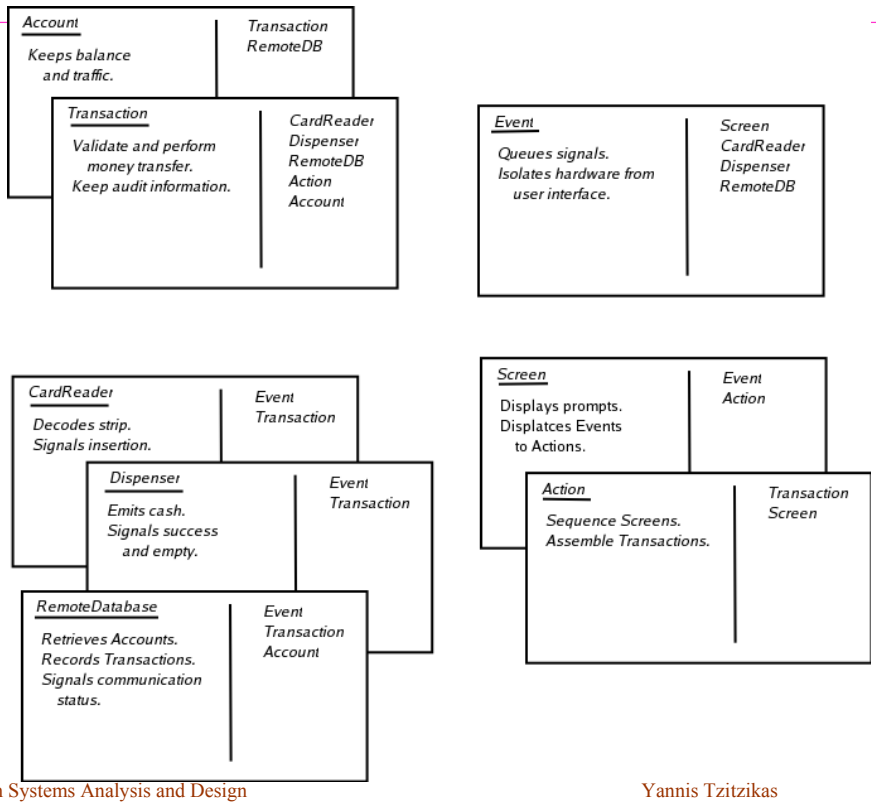
- *This application will support the operations of a technical library for an R&D organization. This includes the searching for and lending of technical library materials, including books, videos, and technical journals. Users will enter their company ids in order to use the system; and they will enter material ID numbers when checking out and returning items.*
- *Each borrower can be lent up to five items. Each type of library item can be lent for a different period of time (books 4 weeks, journals 2 weeks, videos 1 week). If returned after their due date, the library user's organization will be charged a fine, based on the type of item(books $1/day, journals $3/day, videos $5/day).*

Εκτέλεση Σεναρίου (Scenario execution)

Τα σενάρια που περιγράφουν οι Περιπτώσεις Χρήσης μπορούν να χρησιμοποιηθούν σαν ένα είδος σεναρίου (script) για το "παιχνίδι ρόλων" (role-playing method) το οποίο θα μας επιτρέπει να κατανοήσουμε τον τρόπο συνεργασίας των κλάσεων και να εντοπίσουμε λάθη ή παραλείψεις

- Start with scenarios that are part of the systems normal operation first.
- Then consider the exceptional scenarios (e.g. error recover)
- For each scenario decide which class is responsible. The owner of the class then picks up his card
  - When a card is in the air it is an object and can do things.
  - The own announces that he needs to fulfill his responsibility.
  - The responsibility is refined in to smaller tasks if possible. These smaller tasks can be fulfilled by the same object or by interacting with other objects. If no other appropriate class exist, maybe you need to make one. This is the fundamental procedure of the scenario execution.

Περιγραφή ενός μηχανήματος ATM με CRC κάρτες



Τα πλεονεκτήματα της χρήσης καρτών CRC

- CRC cards allow … animated discussion among the team
  - the participants can experience how the system will work
- with CRC cards it is easy and fast to explore various alternatives (sequence diagrams can be slow to draw)
- CRC cards are portable (no computers are required)
- CRC cards are a useful tool for teaching people the object-oriented paradigm.

Ένα σύνολο βημάτων για τη Μοντελοποίηση Δομής με κάρτες CRC

1. Create CRC cards by analyzing the text of the Use Cases
2. Brainstorm additional candidate classes
3. Role-play each use-case using the CRC cards.
4. Create the class diagram based on the CRC cards.
5. Review the structural model for missing and/or unnecessary classes, attributes, operations, and relationships.

Διαγράμματα Κλάσεων
Class Diagrams

Τι είναι Κλάση;

Μια **κλάση** περιγράφει ένα *σύνολο αντικειμένων* με:

- Όμοιες ιδιότητες
  - similar properties (attributes)
- Κοινή συμπεριφορά
  - common behaviour (operations)
- Κοινές συσχετίσεις με άλλα αντικείμενα
  - common associations to other objects.

Πως μπορούμε ορίσουμε τις κλάσεις;

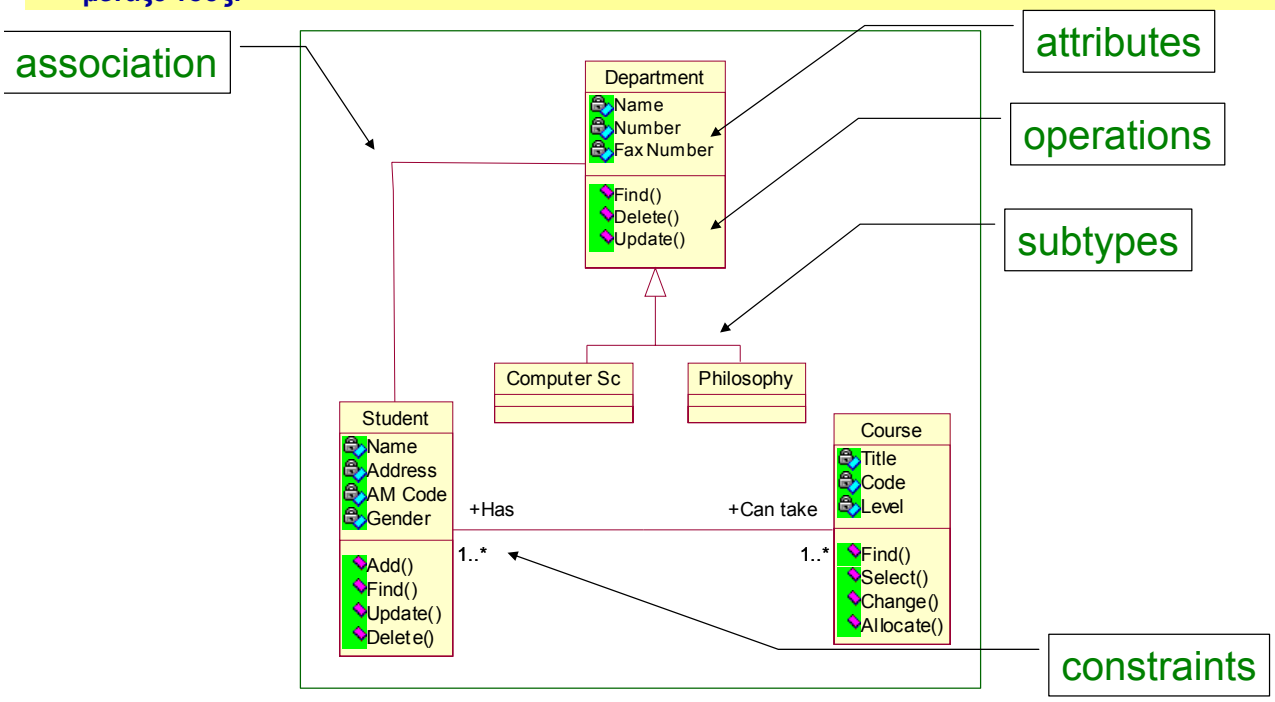
- Χρησιμοποιώντας .. κοινή λογική (common sense) (όταν καταλαβαίνουμε το πεδίο εφαρμογής)
- Ακούγοντας τους ειδικούς του πεδίου εφαρμογής (domain experts)
- Κάνοντας ανάλυση CRC.

Γνωρίσματα, Λειτουργίες και Σχέσεις
(Attributes, Operations and Relationships)

- Γνωρίσματα (Attributes)
  - Μονάδες πληροφορίας που αφορούν στην κλάση ή τα στιγμιότυπα της
  - Αρχικά ορίζουμε μόνο εκείνα που είναι σημαντικά με το πεδίο εφαρμογής (και σχετίζονται με τις απαιτήσεις)
- Λειτουργίες (Operations)
  - Πράξεις/δράσεις που να αντικείμενα μπορούν να κάνουν
  - Αρχικά εστιάζουμε μόνο στις λειτουργίες που αφορούν στις απαιτήσεις.
- Σχέσεις (Relationships)
  - Γενίκευση (Generalization)
    - Επιτρέπει την κληρονόμηση γνωρισμάτων και λειτουργιών
  - Συσσωμάτωση (Aggregation)
    - Σχετίζει τα τμήματα με το όλο
  - Συσχέτιση (Association)
    - Ποικίλες συσχετίσεις μεταξύ κλάσεων

Διάγραμμα Κλάσεων
Class Diagram

Οπτικοποιεί όλα τα προηγούμενα: κλάσεις (όνομα, γνωρίσματα, λειτουργίες) και τις σχέσεις μεταξύ τους.



Ορίζουμε κλάσεις για διάφορα τμήματα του συστήματος

Τυπικά παραδείγματα:

- *Κλάσεις Πεδίου Εφαρμογής (application domain classes)*
- *Κλάσεις Διεπαφής Χρήστη (user interface classes)*
- *Κλάσεις Δομής Δεδομένων (data structure classes)*
- *Κλάσεις Δομής Αρχείων (file structure classes)*
- *…*
- *…*

Οι 3 προοπτικές ενός Διαγράμματος Κλάσεων (the 3 Perspectives of a Class Diagram)

- Θα μπορούσαμε να διακρίνουμε 3 προοπτικές για τη σχεδίαση ενός διαγράμματος κλάσεων (εννοιολογικού μοντέλου γενικότερα)
  - ***Εννοιολογική (Conceptual)***
    - Ανεξάρτητη υλοποίησης. Συχνά αναφέρεται ως **μοντέλο πεδίου** (**domain model).**
  - ***Προδιαγραφής (Specification)***
    - Αφορά κυρίως στις **διεπαφές λογισμικού (interfaces of the software)**, και δεν αντανακλά την υλοποίηση
  - ***Υλοποίησης (Implementation)***
    - Εδώ μοντελοποιούμε τις συγκεκριμένες **κλάσεις υλοποίησης (implementation classes)**.
- Οι παραπάνω προοπτικές (perspectives) τυπικά δεν αποτελούν μέρος της UML
  - By *tagging* classes with a stereotype, we can provide an indication of the perspective

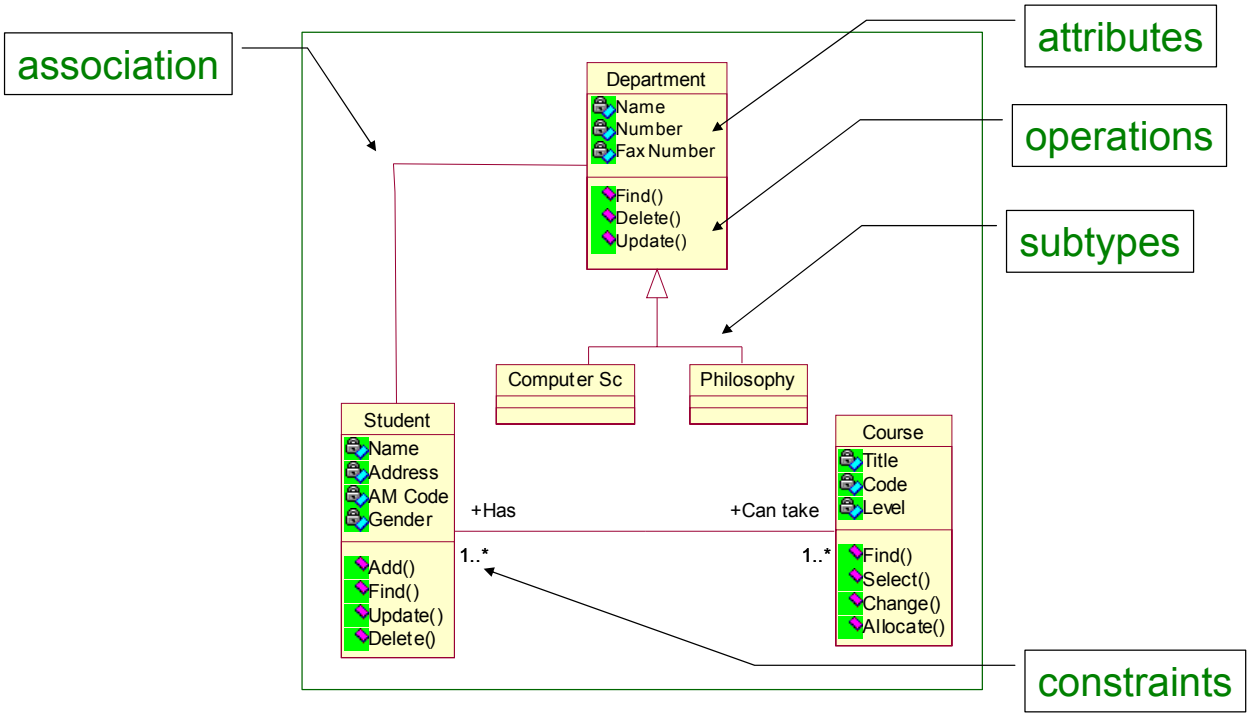
Οι 3 προοπτικές ενός Διαγράμματος Κλάσεων
The 3 Perspectives of a Class Diagram

- There are 3 perspectives for the design of a class diagram (of a conceptual model in general)
  - ***Conceptual***
    - Independent of implementation. This is often called **domain model.**
  - ***Specification***
    - Based on **interfaces of the software**, not the implementation
  - ***Implementation***
    - Here we model the **implementation classes**. This is the most often used perspective
- Perspectives are not part of the formal UML. By *tagging* classes with a stereotype, we can provide an indication of the perspective

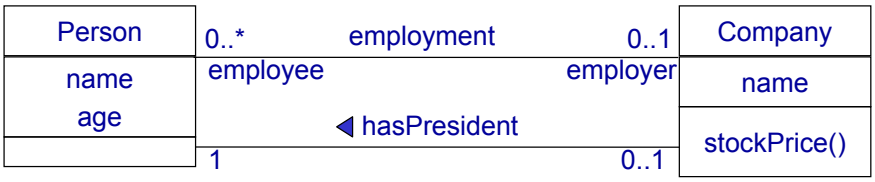
Keywords (UML v2) and Stereotypes (UML v1)

- It is the core extension mechanism of the UML
- If you need a modeling construct that isn't in the UML but is similar to something that is, you treat your construct as a **stereotype** (UML v.1), or **keyword** (UML v.2) of the UML construct.
- Denoted by **<<name>>** (or sometimes **{name}**)
- E.g. interface
  - A UML interface is a class with only public operations with no method bodies nor attributes (like in Java, CORBA)
  - Denoted by <<interface>>
- We could define stereotypes of classes, associations, generalization.
  - We would consider them as subtypes of the meta-model types Class, Association, Generalization

Παράδειγμα διαγράμματος Κλάσεων

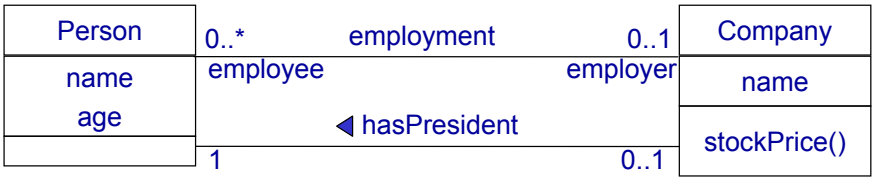


Συσχετίσεις (προοπτική: *Εννοιολογική*)
Associations (Perspective: *Conceptual*)



- Αναπαριστάνουν <u>δυαδικές συσχετίσεις</u> μεταξύ των <u>στιγμιοτύπων των κλάσεων</u>
- Σε κάθε άκρο μπορούμε να δώσουμε ένα όνομα, το <u>όνομα ρόλου</u>

Συσχετίσεις (προοπτική: *Εννοιολογική*)
Associations (Perspective: *Conceptual*)



Περιορισμοί Πολλαπλότητας (Multiplicity constraints)

- Εκφράζουν το πλήθος των αντικειμένων που μπορούν να μετέχουν σε μια συσχέτιση. Συγκεκριμένα εκφράζονται με κάτω και πάνω όρια.

  ***** ≡ **0..*** ≡ **0..** ∞ // κανένας περιορισμός

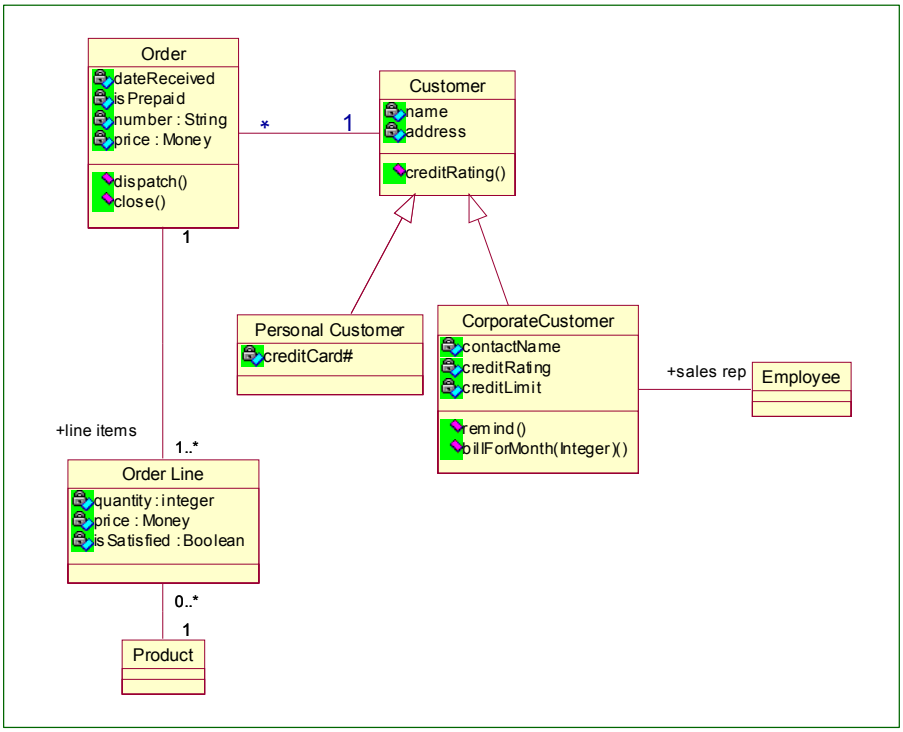
  1 ≡ **1..1** // υποχρεωτική (mandatory) και μονότιμη (single-valued) συσχέτιση

  0..1 // μονότιμη συσχέτιση (προαιρετική)

  1..* // υποχρεωτική και πλειότιμη (multi-valued) συσχέτιση

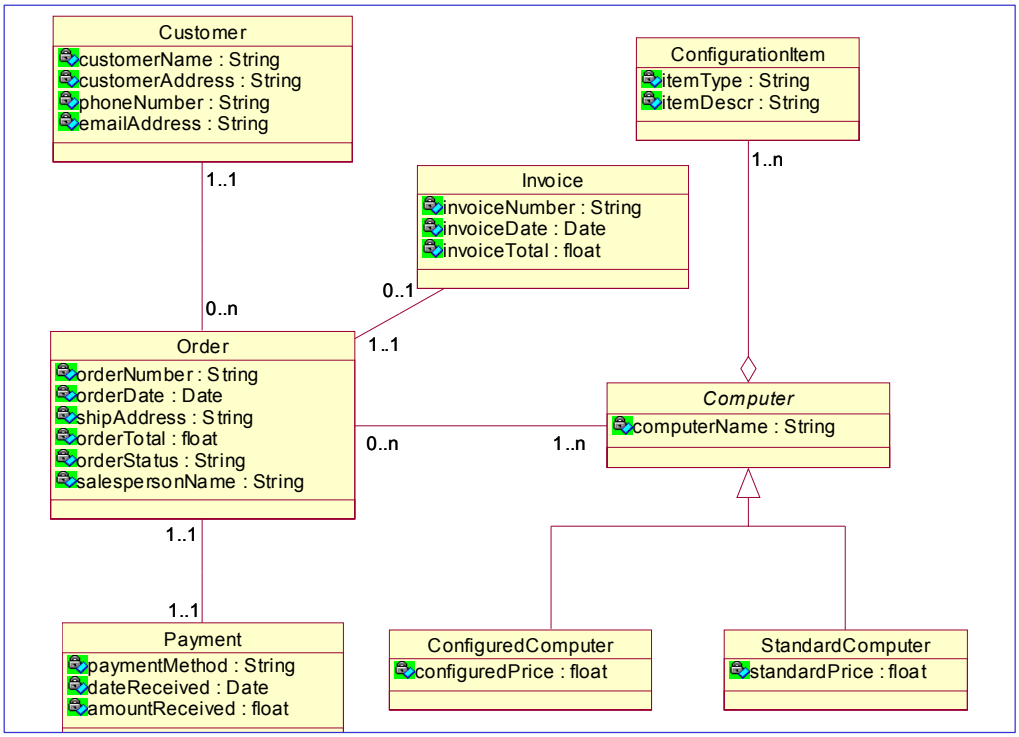
- Παραδείγματα πιο συγκεκριμένων περιορισμών:
  - 1..11 (για ποδοσφαιρικές ομάδες)
  - 3..4 (για τροχούς αυτοκινήτων)

Άλλο ένα παράδειγμα



- Do it on class?

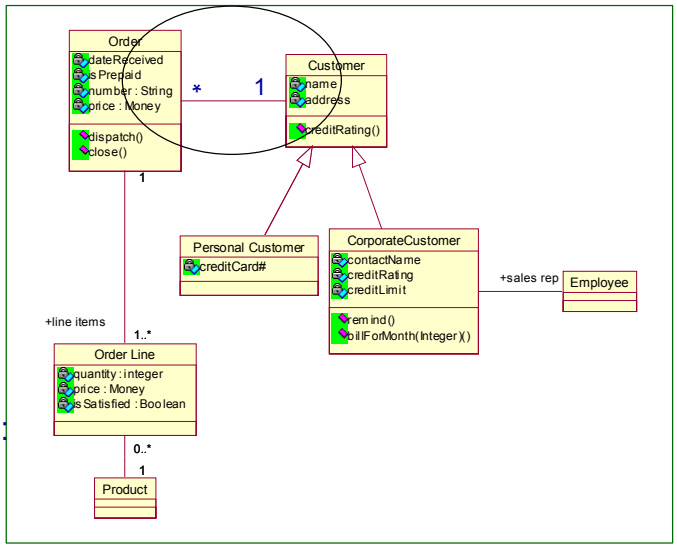
Συσχετίσεις και περιορισμοί πολλαπλότητας (Associations and multiplicity constraints)



- Students should give the mult contraints

Συσχετίσεις (προοπτική: *Προδιαγραφής*) Associations (Perspective: *Specification*)

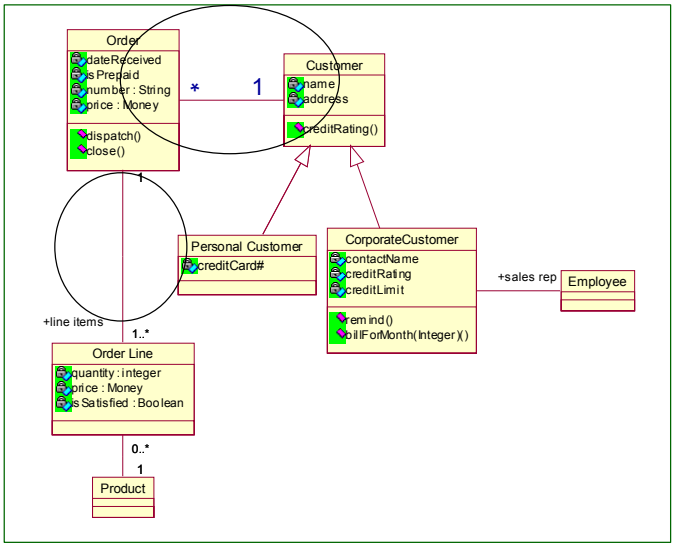
- Here associations represent responsibilities (read & update)
- from this diagram we may say that:
  - there are methods associated with customer that return the orders of a given customer has made
  - the reverse for Order (return the customer)
- we cannot infer implementation details:
  - I.e. if Order class contains a pointer to Customer, or if it calls a method of customer



Συσχετίσεις (προοπτική: *Προδιαγραφή*)
Associations (Perspective: *Specification*)

- If this were an specification model we could infer the following interface for an Order class

```
class Order {
    public Customer  getCustomer();
    public Set       getOrderLines();
```

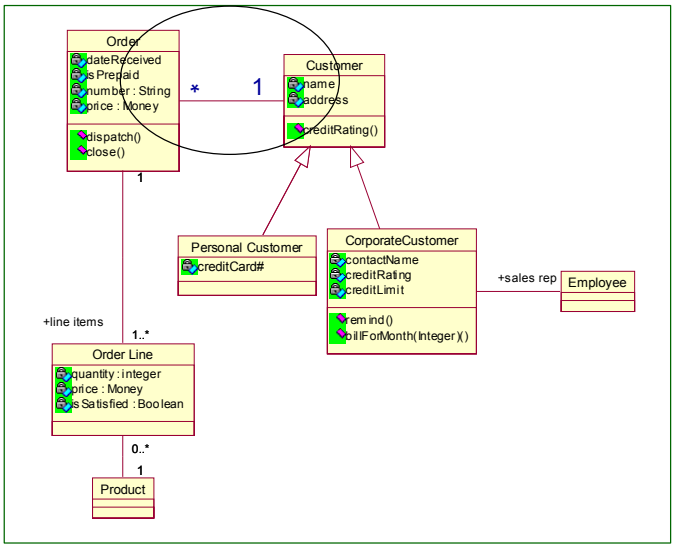


Συσχετίσεις (προοπτική: *Υλοποίηση*)
Associations (Perspective: *Implementation*)

- If this were an implementation model we could infer:

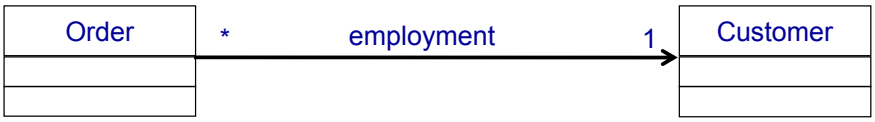
```
class Order {
    private Customer  _customer;
    private Set       _orderLines;

class Customer {
    private Set  _orders;
```

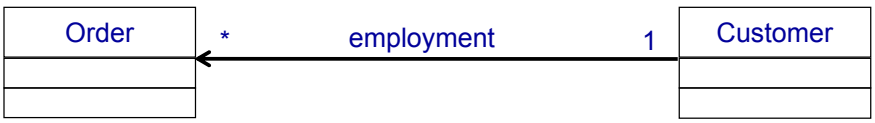


- Η πλοηγησιμότητα είναι χρήσιμη μόνο για την προοπτική *Προδιαγραφής* και *Υλοποίησης* (δεν είναι χρήσιμη για την *Εννοιολογική* προοπτική)

Πλοηγησιμότητα (μονοκατευθυντικές σχέσεις)
Navigability (unidirectional associations)

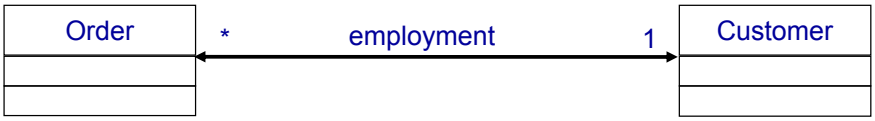


- *Spec*: Order has **responsibility** to tell you which customer it is for
- *Impl*: Order contains a **pointer** to Customer (and not the other way around)



- *Spec*: Customer has **responsibility** to tell you his/her orders
- *Impl*: Customer contains a **set of pointers** to Orders

Navigability (bidirectional associations)

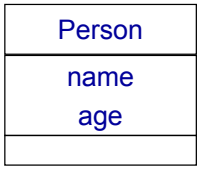


- *Spec*: **Both** have the responsibility to tell you the other end
- *Impl*: **Both** contain pointers to the other end

When we implement a bidirectional association in a programming language we have to be sure that both properties are updated.

note.

Attributes
Γνωρίσματα



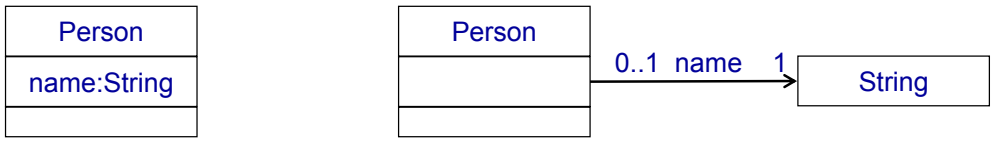
- *Εννοιολογική*: **Ιδιότητα (Property)**
  - e.g. a Person has a name
- *Προδιαγραφής:*
  - e.g. a Person object can tell/set its name
- *Υλοποίηση*:
  - e.g. a Person object has a field (instance variable)

- Like associations
  - small, simple classes, such as strings, dates, money objects, and non-object values like Integer and Real.

Attribute syntax in UML:

visibility name: type = defaultValue

Η διαφορά μεταξύ Γνωρισμάτων και Συσχετίσεων (Difference between Attributes and Associations)



- *Εννοιολογική* προοπτική: Καμία διαφορά
  - attributes are usually single-valued
  - However they can be optional, mandatory, have multiplicity
    - e.g. dateReceived [0..1]: Date
- Προοπτική *Προδιαγραφής/Υλοποίησης;* :
  - attribute => navigability from the type to the attribute range only
  - each person has its own copy of attribute object (value semantics rather than reference semantics)

Λειτουργίες Operations

Είναι οι διαδικασίες που μια κλάση μπορεί να διενεργήσει

Company
name
hire(p:Person) fire(p:Person) promote(p,incr)

- *Εννοιολογική* προοπτική:
  - Indicate the principal responsibilities (described in a couple of words)
- *Προδιαγραφική* προοπτική :
  - Public methods on a type
- *Υλοποιητική* προοπτική :
  - plus private/protected operations

Οι λειτουργίες στην UML
Operations in UML

συντακτικό:

visibility name (parameter-list): return-type-expression {property-string}

- **visibility**:
  - **+** : public (by all used)
  - **-** : private (by owning class)
  - **#** : protected (by owning class and its subclasses)

- **name**: a string

Example: Protected scope in Java

```
package one;

public class A
  {
  protected int p;
  }
```

```
package two;
import one. A;

class B extends A
  {
  void myMethod()
    {
    p = 1;     // ok
    A a = new A();
    a.p = 1;   // not ok, p would have to be public for this to work.
    }
  }
```

Οι λειτουργίες στην UML
Operations in UML

syntax:

visibility name (parameter-list): return-type-expression {property-string}

- **parameter-list**: comma separated parameters with syntax that of attributes (plus direction), i.e. **direction name: type = default value**
  - direction (default: in)
    - **in**: used for input
    - **out**: used for output
    - **inout**: used for both
- **return-type expression**: comma-separated list of return types
    - can be more than one

In Java: methods do not support out/inout parameters for primitives
(are passed by value).

In C: exchange(int a, int b) vs exchange(int *a, int *b)

Direction of Parameters
Example: C++

void f(int a, int &b, const int &c);
a is a value parameter,
b is a reference parameter,
and *c* is a const-reference parameter.

- When a parameter is passed by **value**, a *copy* of the parameter is made.
- When a parameter is passed by **reference**, conceptually, the actual parameter itself is passed (and just given a new name -- the name of the corresponding formal parameter). Therefore, any changes made to the formal parameter *do* affect the actual parameter.
- Another reason to use reference parameters is when you don't want the function to modify an actual parameter, but the actual parameter is very large, and you want to avoid the overhead of creating a copy. Of course, this only works if the function does not modify its formal parameter. To be sure that the actual parameter is not "accidentally" modified, you should use a **const-reference** parameter. Declaring the parameter to be *const* tells the compiler that it should not be changed; if the function does change the parameter, you will get a compile-time warning (possibly an error on some systems).

For more about parameter passing in C++ see:
http://www.cs.wisc.edu/~hasti/cs368/CppTutorial/NOTES/PARAMS.html

Direction of Parameters Example: C#

- **Value parameters:**
  value parameter is also called **In** parameter. A parameter declared with no modifiers is a value parameter.
- **Reference parameters:**
  A parameter declared with a **ref** modifier is a reference parameter.
- **Output parameters:**
  A parameter declared with an **out** modifier is an output parameter. Similar to a reference parameter, an output parameter does not create a new storage location. Instead, an output parameter represents the same storage location as the variable given as the argument in the method invocation. Every output parameter of a method must be definitely assigned before the method returns. The only differences with reference parameters are:
  - The variable specified on the invocation doesn't need to have been assigned a value before it is passed to the function member. If the function member completes normally, the variable is considered to be assigned afterwards (so you can then "read" it).
  - The parameter is considered initially unassigned (in other words, you must assign it a value before you can "read" it in the function member).
  - The parameter *must* be assigned a value before the function member completes normally.

For more see: http://www.yoda.arachsys.com/csharp/parameters.html

Οι λειτουργίες στην UML Operations in UML

syntax:

visibility name (parameter-list): return-type-expression {property-string}

- **property-string**: property values that apply to the given operation
  - {abstract}: it requires a child to complete the implementation
  - {leaf}: not polymorphic (may not be overridden) // like *final* in Java
  - {query}: the execution of the operation leaves the state of the system unchanged
  - {sequential}: only one flow should be in the object at a time
  - {guarded}:
  - {concurrent}
  - {static}: it behaves as a global procedure

Οι λειτουργίες στην UML
Operations in UML

syntax:

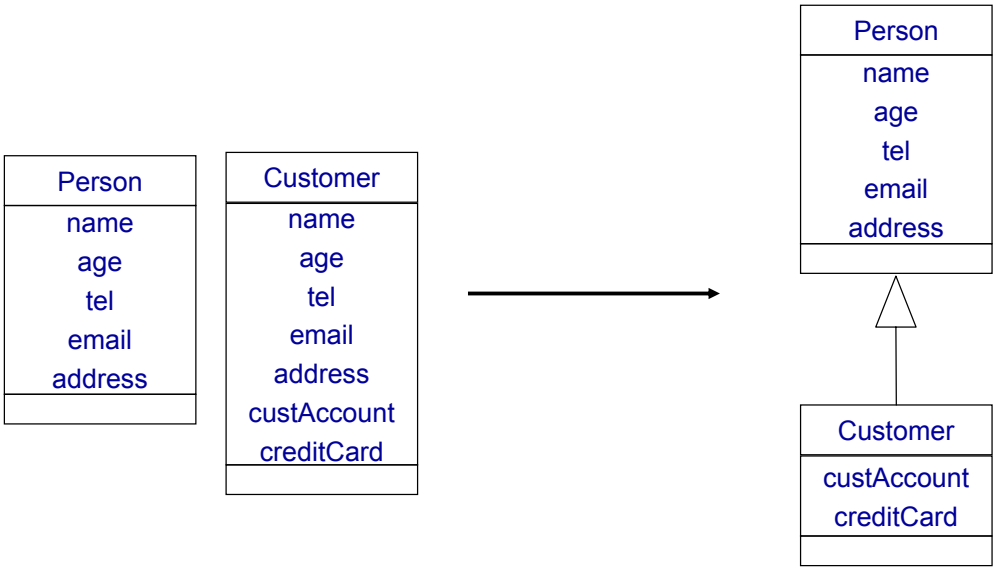
visibility name (parameter-list): return-type-expression {property-string}

- Examples:
  - +balanceOn(date:Date):Money

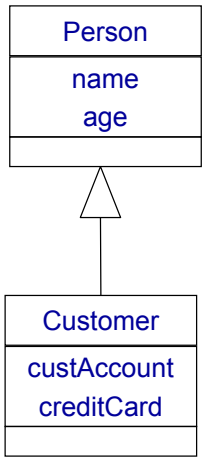
Λειτουργίες (II)
Operations (II)

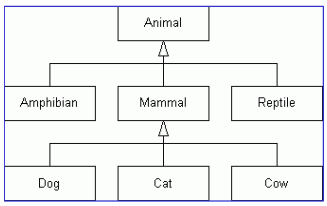
- Constructor
  - creates an object
- Queries vs Modifiers
  - query: an operation that gets a value from a class without changing its state (i.e. without side effects)
    - we mark them with the constraint {query}
  - modifier: an operation that changes the state
- Operations vs Methods
  - **operation**: the procedure call (else called method call or method declaration)
  - **method**: the body of the procedure (else called method body)
  - the above are different if we have polymorphism
    - if we have a supertype and three subtypes, each of which overrides the supertype's "foo" operation, then we have 1 operation and 4 methods that implement it.

Γενίκευση
Generalization



Γενίκευση
Generalization





- *Εννοιολογική* προοπτική:
  - Subset of instances
  - inheritance of properties
- *Προδιαγραφική* προοπτική:
  - The interface of the subtype must include all elements from the interface of the supertype.
  - The subtype's interface is said to "conform to" the sypertype interface
- *Υλοποιητική* προοπτική:
  - Associated with inheritance in PLs
  - Subtypes inherit all methods and fields and may override inherited methods

Περιορισμοί
Constraint Rules

- A diagram actually specifies a set of constraints
- However, we need to express more constraints (apart from those we have seen so far)
- UML wants to put them inside braces { } // e.g. informal English
- There is also a formal **Object Constraint Language** (OCL)
  - Warmer/Kleppe 98. OCL will be covered in a subsequent lecture.
- Ideally, they should be implemented by *assertions* in the PL
- These correspond with the “Design by Contract” notion of invariants.

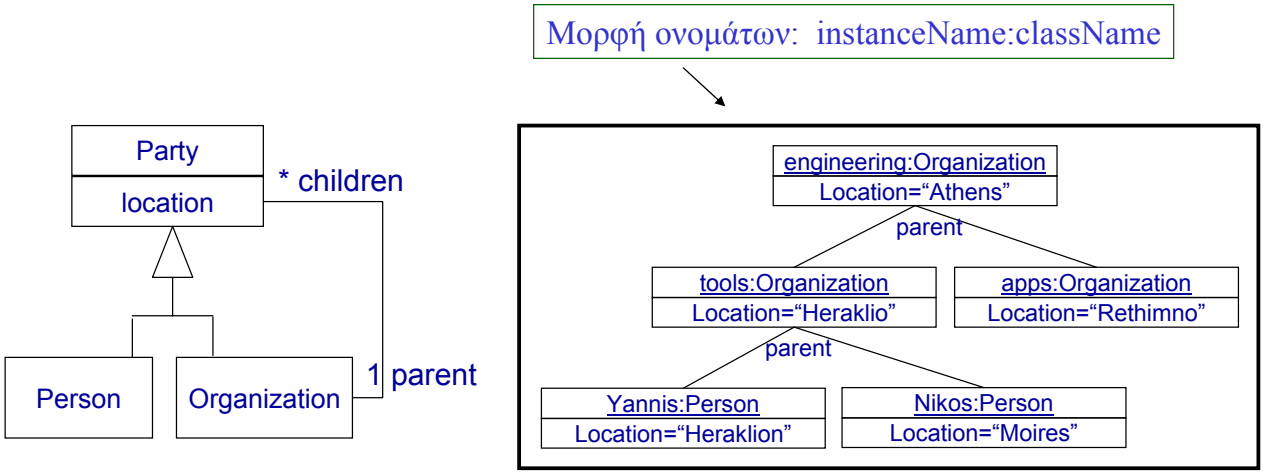
More issues on class diagrams will be covered in the next lecture.

Object Diagrams

Διαγράμματα Αντικειμένων

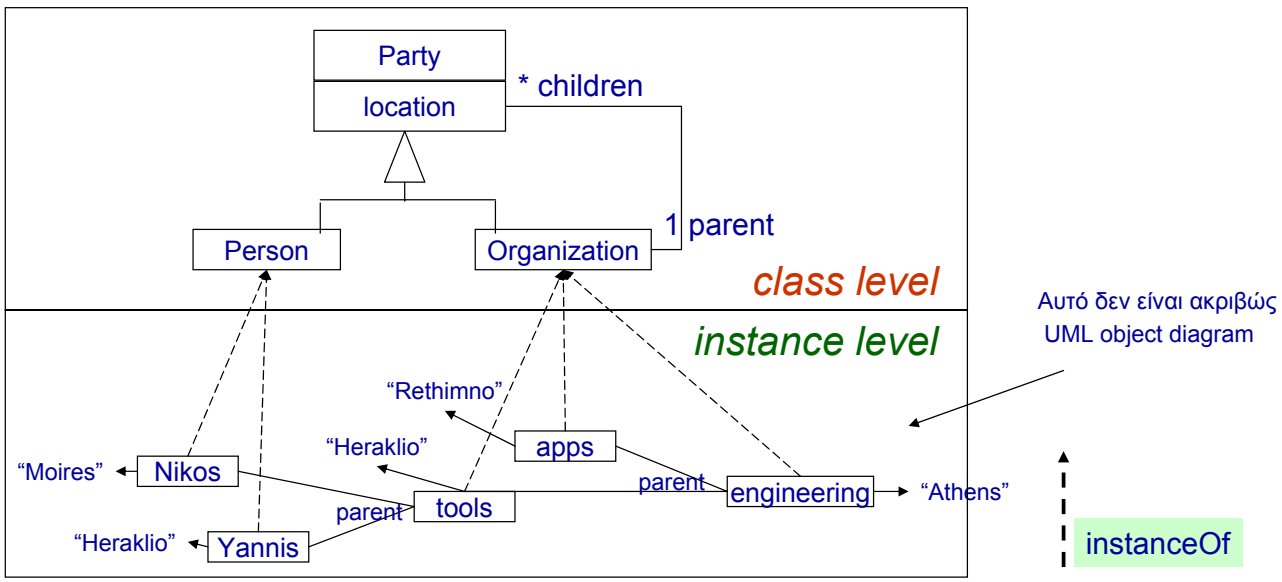
Διάγραμμα Αντικειμένων
Object Diagrams

- Παρουσιάζει τα στιγμιότυπα και όχι τις κλάσεις. Αλλιώς λέγονται διαγράμματα στιγμιότυπων (instance diagrams)
- Μπορούν να δείξουν παραδείγματα των τρόπων διασύνδεσης των αντικειμένων
  - (it is like a collaboration diagram but without messages)



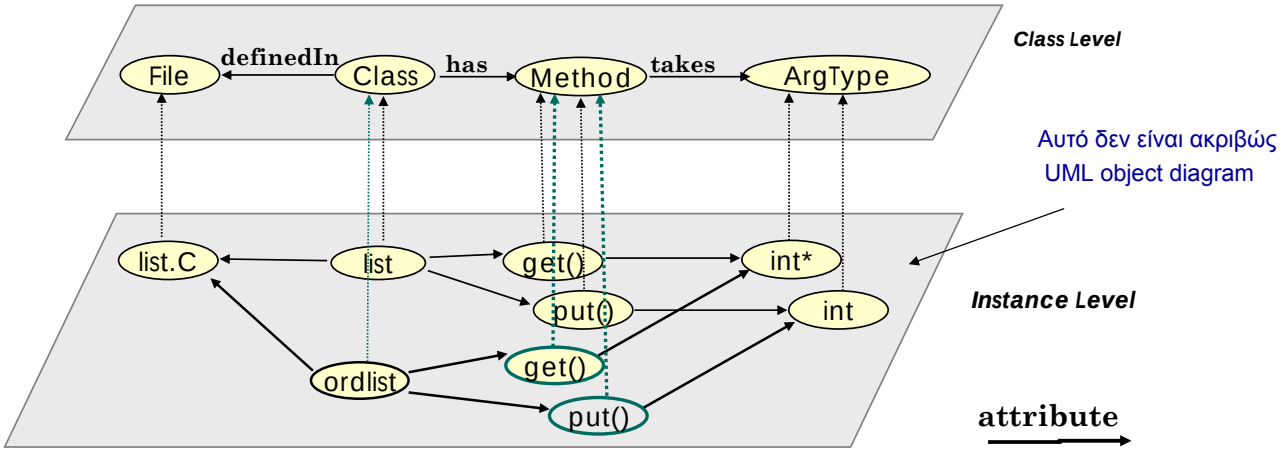
Παρουσιάζοντας ένα Διάγραμμα Κλάσεων και Αντικειμένων μαζί
Presenting Class and Object Diagrams Together

- Καμιά φορά κάτι τέτοιο είναι χρήσιμο (και όταν το διάγραμμα των κλάσεων δεν είναι πολύ μεγάλο)

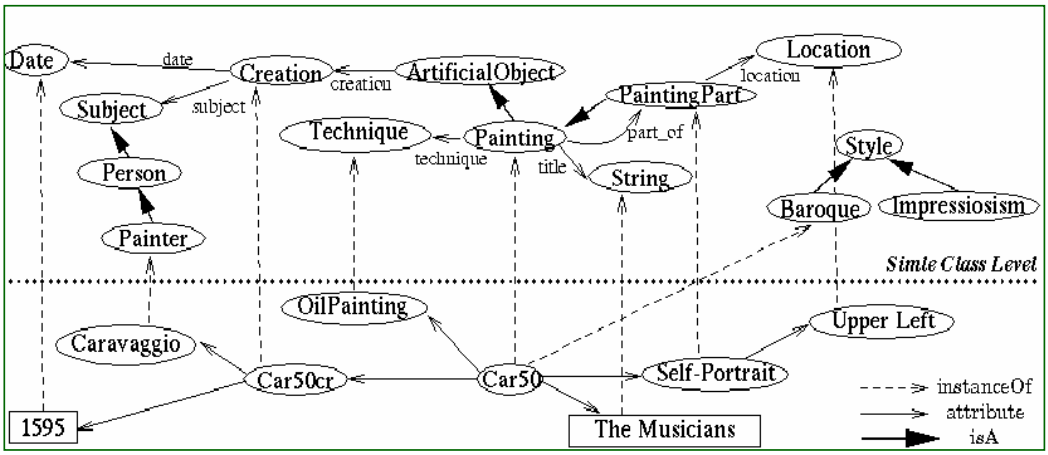


Παρουσιάζοντας ένα Διάγραμμα Κλάσεων και Αντικειμένων μαζί
Presenting Class and Object Diagrams Together

- Παράδειγμα ενός μοντέλου για τη στατική ανάλυση κώδικα



Παρουσιάζοντας ένα Διάγραμμα Κλάσεων και Αντικειμένων μαζί
Presenting Class and Object Diagrams Together



Πρακτικές Συμβουλές

- Αν είστε αναλυτές => ζωγραφίστε εννοιολογικά μοντέλα
- Αν είστε προγραμματιστές => επικεντρωθείτε στα μοντέλα προδιαγραφής
- Ζωγραφίστε μοντέλα υλοποίησης μόνο αν θέλετε να εικονογραφήσετε μια συγκεκριμένη τεχνική υλοποίησης (π.χ. Patterns)
- Μην ζωγραφίζετε διαγράμματα για τα πάντα. Συγκεντρωθείτε στα σημαντικές πλευρές του προβλήματος

Παράδειγμα

- Φτιάξτε το διάγραμμα κλάσεων για το:

ΥΠΟΥΡΓΕΙΟ ΕΘΝΙΚΗΣ ΠΑΙΔΕΙΑΣ ΚΑΙ ΘΡΗΣΚΕΥΜΑΤΩΝ
ΚΕΝΤΡΟ ΕΛΛΗΝΙΚΗΣ ΓΛΩΣΣΑΣ
Τμήμα Στήριξης και Προβολής της ελληνικής Γλώσσας

ΔΕΛΤΙΟ ΠΑΡΑΓΓΕΛΙΑΣ

ΗΜΕΡΟΜΗΝΙΑ:	
ΟΝΟΜΑΤΕΠΩΝΥΜΟ:	
ΔΙΕΥΘΥΝΣΗ:	
ΠΟΛΗ: ΤΑΧ. ΚΩΔΙΚΑΣ:	
ΧΩΡΑ:	
ΑΦΜ: ΔΟΥ:	
ΤΗΛΕΦΩΝΟ: FAX:	
E-MAIL:	

ΤΙΤΛΟΣ	ΕΠΙΠΕΔΟ	ΠΟΣΟΤΗΤΑ	ΤΙΜΗ
ΣΥΝΟΛΟ			

ΣΗΜ: ΓΙΑ ΤΗΝ ΠΑΡΑΓΓΕΛΙΑ ΤΩΝ ΕΞΕΤΑΣΤΙΚΩΝ ΘΕΜΑΤΩΝ ΤΩΝ ΠΡΟΗΓΟΥΜΕΝΩΝ ΕΤΩΝ Ή ΤΩΝ ΘΕΜΑΤΩΝ ΓΙΑ ΕΞΑΣΚΗΣΗ ΠΡΕΠΕΙ ΝΑ ΔΙΕΥΚΡΙΝΙΖΕΤΕ ΤΟ ΕΠΙΠΕΔΟ ΠΟΥ ΕΠΙΘΥΜΕΙΤΕ ΝΑ ΠΡΟΜΗΘΕΥΤΕΙΤΕ.

Συχνά λάθη

- Οι συσχετίσεις απεικονίζονται και ως ακμές που συνδέουν τους κόμβους των κλάσεων και ως γνωρίσματα (στο εσωτερικό των κόμβων)
  - Τα δεύτερα είναι η υλοποίηση των συσχετίσεων (άρα δεν έχει νόημα να εμφανίζονται στο ίδιο διάγραμμα)

Σύνοψη

Η δομική μοντελοποίηση είναι ίσως η πιο καθοριστική όψη της μοντελοποίησης αφού καθορίζει σε μεγάλο βαθμό αρκετά από τα επόμενα στάδια και απόψεις μοντελοποίησης. Για παράδειγμα, καθορίζει τα προς αποθήκευση δεδομένα, τον τρόπο υλοποίησης της συμπεριφοράς (τρόπο υλοποίησης των λειτουργιών του συστήματος) κα. Η μοντελοποίηση δομής μπορεί να γίνει σταδιακά. Αρχικά προσπαθούμε να εντοπίσουμε τις πιο σημαντικές έννοιες του πεδίου εφαρμογής. Τις δηλώνουμε ως κλάσεις, προσθέτουμε μερικά από τα βασικά τους γνωρίσματα και κατόπιν ορίζουμε τις συσχετίσεις μεταξύ των κλάσεων. Το διάγραμμα που θα προκύψει πρέπει να είναι κατανοητό από έναν ειδικό του πεδίου εφαρμογής (πελάτη) και να μην σχετίζεται με θέματα υλοποίησης ή άλλα τεχνικά θέματα. Άρα μπορούμε να το δημιουργήσουμε ακόμα και κατά τη διάρκεια των συζητήσεων μας με τον πελάτη προκειμένου να (α) βεβαιωθούμε ότι έχουμε κατανοήσει και μοντελοποιήσει ορθά τον κόσμο της εφαρμογής, (β) ζητήσουμε διευκρινήσεις και επιπλέον λεπτομέρειες για κάποια σημεία του, (γ) έχουμε ένα κοινό σημείο αναφοράς και συμφωνίας με τον πελάτη

Σύνοψη (ΙΙ)

Μετά από αυτό το στάδιο μπορούμε να συμπληρώσουμε και να εκλεπτύνουμε την περιγραφή των κλάσεων και των συσχετίσεών τους (προσθέτοντας γνωρίσματα, λειτουργίες, περιορισμούς).

Αξίζει εδώ να σημειωθεί ότι η δήλωση των κλάσεων με τη γραφική γλώσσα της UML μας επιτρέπει να έχουμε μια καλή εποπτική εικόνα του σχεδίου η οποία μας επιτρέπει να συλλογιστούμε και να ελέγξουμε το βαθμό στον οποίο το διάγραμμα καλύπτει τις απαιτήσεις, να διερευνήσουμε γρήγορα εναλλακτικούς τρόπους μοντελοποίησης, να μελετήσουμε ζητήματα πληρότητας, να εντοπίσουμε λάθη, να προσθέσουμε περιορισμούς ακεραιότητας κλπ.

Αν η σύνταξη του γίνεται με ένα εργαλείο CASE η διαδικασία αυτή μπορεί να γίνει πολύ γρήγορα (αρκεί να έχουμε εξοικειωθεί με το εργαλείο). Η άμεση συγγραφή κώδικα σε αρχεία κειμένου δεν έχει τα παραπάνω πλεονεκτήματα. Οι αλλαγές είναι επίσης γρηγορότερες σε ένα γραφικό περιβάλλον απ' ότι σε κείμενα.

Στα επόμενα στάδια μπορούμε να εξειδικεύσουμε το διάγραμμα κλάσεων βάσει της γλώσσας προγραμματισμού που θα χρησιμοποιηθεί στην υλοποίηση. Η προσθήκη περιορισμών εκφρασμένων σε OCL μπορεί να κάνει ακόμα λεπτομερέστερο το σχέδιο και να το φέρει πιο κοντά στην τελική υλοποίηση.

Περίληψη

- *Οι κάρτες CRC μας βοηθούν να συλλάβουμε τα ουσιώδη συστατικά των κλάσεων.*
- *Τα διαγράμματα κλάσεων και αντικειμένων μπορούν να μας δείξουν την δομή ενός αντικειμενοστρεφούς συστήματος.*
- *Η κατασκευή δομικών μοντέλων είναι μια επαναληπτική διαδικασία η οποία περιλαμβάνει: ανάλυση κειμένων, ανταλλαγή ιδεών (και "παιχνίδι ρόλων») και δημιουργία διαγραμμάτων.*

Πηγές

- **Systems Analysis and Design with UML Version 2.0** (2nd edition) by A. Dennis, B. Haley Wixom, D. Tegarden, Wiley, 2005. CHAPTER 7
- **UML Distilled: A Brief Guide to the Standard Object Modeling Language** (3rd Edition) by Martin Fowler, Addison Wesley, 2004. Chap. 3
- **The Unified Modeling Language User Guide** (2nd edition) by G. Booch, J. Rumbaugh, I. Jacobson, Addison Wesley, 2004, Chap 8 (advanced: 9-10)

- CRC cards:A tutorial regarding CRC cards can be found at:
  - *http://www.csc.calpoly.edu/~dbutler/tutorials/winter96/crc_b/*