DIS501 Ανάλυση και Σχεδίαση Πληροφοριακών Συστημάτων

Lecture 5: Εισαγωγή στην Αντικειμενοστρεφή προσέγγιση

BY: ΔΡ ΑΥΓΟΥΣΤΑ ΚΥΡΙΑΚΙΔΟΥ ΖΑΧΑΡΟΥΔΙΟΥ

ΠΩΣ ΣΧΕΔΙΑΖΟΥΜΕ/ΜΟΝΤΕΛΟΠΟΙΟΥΜΕ ΤΟ ΠΡΟΙΟΝ?

Συμβατικές απαντήσεις από την SSAD (Δομημένη Ανάλυση & Σχεδίαση - Structured Systems Analysis and Design)

- Μοντέλο διαδικασίας Process model - το διάγραμμα ροής δεδομένων και το minispec (Data flow diagrams)
- Μοντέλο δεδομένων Data model - διαγράμματα οντοτήτων – συσχετίσεων (Entity Reslationship Diagrams)
- Μοντέλο συμπεριφοράς - entity life history

Each model taken separately and with an orderly transition (decomposition)

Πιο μοντέρνες απαντήσεις από το OOAD (Αντικειμενοστραφής Ανάλυση & Σχεδίαση - Object-Oriented Analysis and Design)

- Μοντέλο περίπτωσης χρήσης – Use case diagrams
- Μοντέλο Αντικειμένων (Object model) – Class diagrams
- Δυναμικό Μοντέλο – Object sequence diagrams, State chart diagrams
- Διάγραμμα Κλάσεων Σχεδιασμού - Design Class Diagram
- Περιπτώσεις χρήσης, κλάσεις και αντικείμενα, χαρακτηριστικά, μεταβίβαση μηνυμάτων και συμπεριφορά. Use Cases, Classes and Objects, attributes, message passing and behaviour

όλα σε ένα ολοκληρωμένο μοντέλο, η μοντελοποίηση γίνεται με επαναληπτικό και σταδιακό τρόπο, υποστηρίζοντας την αφαίρετικότητα (abstraction) και όχι την αποσύνθεση(***decomposition)***

OOAD VS SSAD

Διαφέρουν στον τρόπο μοντελοποίησης και υλοποίησης του συστήματος (πώς βλέπουμε το σύστημα και πώς το οργανώνουμε**):**

- Structured Systems Analysis & Design **(Σκέψου το σαν μια συνταγή μαγειρικής 📜)**
  - Σχεδιάζουμε το σύστημα βήμα-βήμα, σαν μια συνταγή που ακολουθεί συγκεκριμένες διαδικασίες.
  - Το σύστημα **χωρίζεται** σε διαδικασίες/processes (τι κάνει) και δεδομένα/data (τι αποθηκεύει).
  - Χρησιμοποιεί διαγράμματα ροής για να δείξει πώς μετακινούνται τα δεδομένα και πώς γίνονται οι λειτουργίες (ροή πληροφοριών (data flow).
  - Εξελίχθηκαν από έναν κύκλο ζωής γραμμικών έργων (linear projects) – αποσυνδεδεμένα από το περιβάλλον τους
    - Waterfall model idea at heart
    - a **data-oriented approach** στην μοντελοποίηση
    - Καλύτερο για παραδοσιακά, στατικά συστήματα που έχουν σαφώς ορισμένες διαδικασίες.
      - π.χ database-driven systems, τραπεζικά συστήματα και λογιστικά προγράμματα, large bureaucratic government systems, critical systems

Madeira Cake

Ingredients

175g/6oz butter, at room temperature

175g/6oz caster sugar

3 free-range eggs

250g/9oz self-raising flour

2-3 tbsp milk

1 lemon, zest only

1-2 thin pieces of candied citron or lemon peel, to decorate

Preparation method

1. Pre-heat the oven to 180C/350F/Gas 4. Grease an 18cm/7in round cake tin, line the base with greaseproof paper and grease the paper.
2. Cream the butter and sugar together in a bowl until pale and fluffy. Beat in the eggs, one at a time, beating the mixture well between each one and adding a tablespoon of the flour with the last egg to prevent the mixture curdling.

Technique: Creaming butter by hand

3. Sift the flour and gently fold in, with enough milk to give a mixture that falls slowly from the spoon. Fold in the lemon zest.

Technique: Zesting citrus fruit

4. Spoon the mixture into the prepared tin and lightly level the top. Bake on the middle shelf of the oven for 30-40 minutes, or until golden-brown on top and a skewer inserted into the centre comes out clean.
5. Remove from the oven and set aside to cool in the tin for 10 minutes, then turn it out on to a wire rack and leave to cool completely.
6. To serve, decorate the cake with the candied peel.

OOAD VS SSAD

Διαφέρουν στον τρόπο μοντελοποίησης και υλοποίησης του συστήματος (πώς βλέπουμε το σύστημα και πώς το οργανώνουμε):

- Object Oriented Analysis & Design (Σκέψου το σαν LEGO 🧱):
  - βλέπει το σύστημα ως μια συλλογή αλληλοεπιδρώντων αντικειμένων - **objects** που έχουν τα δικά τους χαρακτηριστικά-δεδομένα (attributes) και τη δική τους λειτουργία/συμπεριφορά (methods)
  - Δίνει έμφαση στο **encapsulation (βάζει πίσω μαζί τα data και το behaviour/processes)** και στην **επαναχρησιμοποίηση - reusability**
    - **Ιδανικό για σύγχρονες εφαρμογές όπως παιχνίδια, εφαρμογές κινητών, τεχνητή νοημοσύνη.**

ΣΥΓΚΡΙΣΗ ΜΕ ΕΝΑ ΠΑΡΑΔΕΙΓΜΑ 🚙

Σχεδιάζουμε ένα **σύστημα για ένα αυτοκίνητο**

📝 **Με SSAD:**

- Θα το σπάσουμε σε διαδικασίες και ροή δεδομένων: **"Ενεργοποίησε κινητήρα" → "Έλεγξε καύσιμα" → "Προώθησε κίνηση" κτλ.**

📌 **Όλα αυτά καταγράφονται με ένα Διάγραμμα Ροής Δεδομένων (DFD), που δείχνει τι συμβαίνει πρώτο, τι δεύτερο κ.λπ.**

🧱 **Με OOAD:**

- Το σύστημα θα αποτελείται από **αντικείμενα** που συνεργάζονται σαν ένα lego όπως:
  - **Αυτοκίνητο** 🚗 (Χαρακτηριστικά: χρώμα, μάρκα, μοντέλο – Λειτουργίες: Επιτάχυνση, επιβράδυνση, αλλαγή ταχύτητας)
  - **Κινητήρας** 🛠️ (Χαρακτηριστικά: Ισχύς, τύπος καυσίμου– Λειτουργίες: Ξεκινά, σταματά)
  - **Τροχοί** 🛞 (Χαρακτηριστικά: Μέγεθος, υλικό – Λειτουργίες: Περιστροφή)
  - **Οδηγός** 👨‍✈️ (Χαρακτηριστικά: Όνομα, ηλικία, επίπεδο εμπειρίας – Λειτουργίες: Οδηγεί)
- Κάθε αντικείμενο ξέρει **τι να κάνει από μόνο του και συνεργάζονται μεταξύ τους.**
- Όταν το **οδηγός** δώσει εντολή για εκκίνηση, το **αυτοκίνητο** καλεί τη μέθοδο **"ξεκινά"** του **κινητήρα**.
- Ο **κινητήρας** στέλνει μήνυα στους **τροχούς**, που αρχίζουν να περιστρέφονται.

ΣΥΓΚΡΙΣΗ SSAD VS OOAD ΣΤΟ ΑΥΤΟΚΊΝΗΤΟ 🚗

	SSAD (Δομημένη)	**OOAD** (Αντικειμενοστρεφές)
Προσέγγιση	Σκέφτεται το σύστημα ως **μια σειρά από βήματα**	Σκέφτεται το σύστημα ως **ένα σύνολο αντικειμένων που συνεργάζονται**
Δομή	Χωρίζεται σε **διαδικασίες και ροές δεδομένων**	Χωρίζεται σε **αντικείμενα με ιδιότητες και λειτουργίες**
Αλλαγές & Επεκτασιμότητα	Δύσκολο να αλλάξει – αν αλλάξει κάτι, πρέπει να ξανασχεδιάσουμε όλο το σύστημα	Εύκολο να αλλάξει – μπορούμε να αντικαταστήσουμε αντικείμενα χωρίς να πειράξουμε τα υπόλοιπα
Παράδειγμα	"Γύρνα το κλειδί" → "Έλεγξε καύσιμα" → "Ενεργοποίησε κινητήρα" → "Κινήσου"	"Οδηγός" ζητά από το "Αυτοκίνητο" να κινηθεί → Το "Αυτοκίνητο" ζητά από τον "Κινητήρα" να ξεκινήσει → Οι "Τροχοί" περιστρέφονται

INTRODUCING THE OO APPROACH

OOAD: Μια τεχνική για την ανάλυση, τον προσδιορισμό και τη μοντελοποίηση των λειτουργικών απαιτήσεων ενός συστήματος.

Object-oriented Analysis

- Focus is on understanding the **problem -** Έμφαση δίνεται στην κατανόηση του προβλήματος
- **What** the system does - **Τι** κάνει το σύστημα
- **Logical (abstract model)**: πτυχές του συστήματος που μπορούν να σχεδιαστούν χωρίς γνώση της πλατφόρμας υλοποίησης (implementation platform)
- Διαδικασία διερεύνησης και μοντελοποίησης των απαιτήσεων

INTRODUCING THE OO APPROACH

Object oriented Design

- Focus is on understanding the **solution - Έμφαση δίνεται στην κατανόηση της λύσης**
- **How** the system does it - **Πώς** το κάνει το σύστημα
- **Physical**: πτυχές του συστήματος που εξαρτώνται από την πλατφόρμα υλοποίησης (implementation platform) που θα χρησιμοποιηθεί
- Επεξεργάζεται και βελτιώνει περεταίρω τα μοντέλα/διαγράμματα ανάλυσης και τα μετατρέπει σε μοντέλα σχεδιασμού προκειμένου να υλοποιηθούν οι απαιτήσεις.

Object oriented Programming: μετατρέπει τα μοντέλα σχεδιασμού σε κώδικα.

OBJECT ORIENTED ANALYSIS AND DESIGN

- **Περισσότερα από 30 χρόνια ιστορίας της ΟΟ ανάλυσης (τέλη της δεκαετίας του 1980)** (μεγαλύτερη ιστορία της ΟΟ σχεδίασης και του προγραμματισμού).
  - Υπήρχε ανάγκη για πιο συνεκτικό τρόπο περιγραφής συστημάτων που συνέδεαν στενά τα δεδομένα (data) και συμπεριφορά (behaviour/processes) σε ένα αντικείμενο (object).

- Προσπάθεια να επανασυνδεθούν τα **data, process and behaviour** ξανά.
  - Κρατώντας τα τρία στοιχεία μαζί, είναι ευκολότερο να μοντελοποιηθεί το πραγματικό πρόβλημα.
  - Η ανάλυση (OOA), η σχεδίαση (OOD) και ο προγραμματισμός (OOP) στις προσεγγίσεις ΟΟ συγχωνεύονται μεταξύ τους. → Καλύτερη επικοινωνία μεταξύ των developers.

Σε αντίθεση με το SSAD που κρατάει τα δεδομένα (data) και το behaviour (process) χωριστά (χρησιμοποιώντας διαγράμματα ροής δεδομένων για τη μοντελοποίηση του behaviour/process και ERDs για τη μοντελοποίηση των δεδομένων (data).

Όχι πλέον, όπως στην SSAD, μια προσέγγιση από πάνω προς τα κάτω, διαίρει και βασίλευε (top-down, divide-and-conquer), αλλά διατηρώντας αυτά τα βασικά στοιχεία μαζί με έναν τρόπο που είναι πιο κοντά στη μοντελοποίηση του πραγματικού προβλήματος.

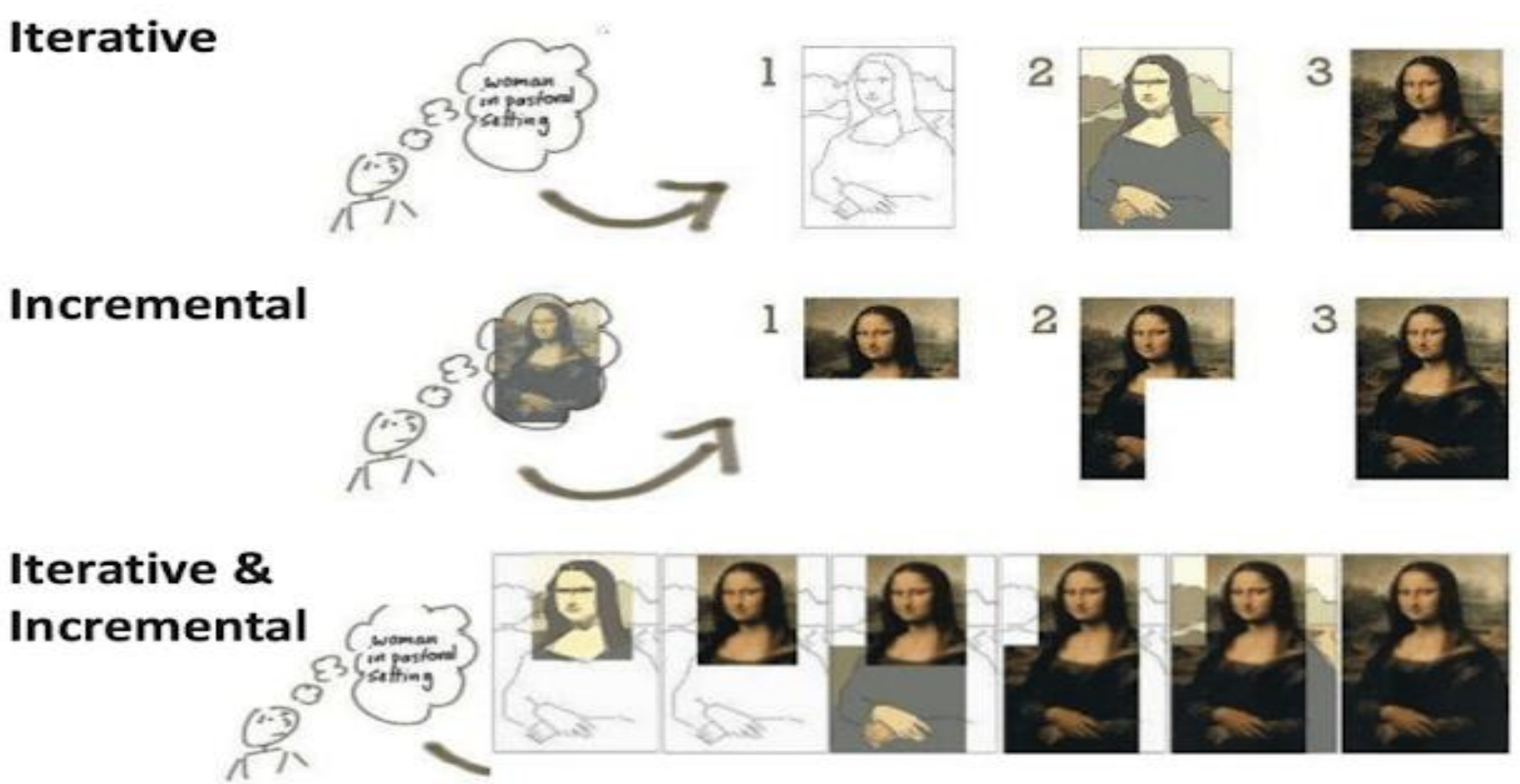
OBJECT ORIENTED ANALYSIS AND DESIGN

- Το πρόβλημα αντιμετωπίζεται ως αντικείμενα (πράγματα) **objects (things) with certain characteristics** με συγκεκριμένα χαρακτηριστικά που αλληλεπιδρούν μεταξύ τους.
- Το σύστημα διαμορφώνεται με τη μορφή ιεραρχιών αντικειμένων (objects) που μοιράζονται/κληροδοτούν γενικά χαρακτηριστικά, αλλά αναγνωρίζουν και τα ιδιαίτερα χαρακτηριστικά του κάθε αντικειμένου.

- Παρέχει ένα καθαρότερο και καλά ολοκληρωμένο μοντέλο της συμπεριφοράς του συστήματος.

- **Εξομαλύνει τη μετάβαση μεταξύ ανάλυσης, σχεδίασης και προγραμματισμού.**
  - Υποστηρίζει την επαναληπτική (επαναλαμβανόμενοι κύκλοι) και την αυξητική(σε μικρότερα τμήματα κάθε φορά) διαδικασία ανάπτυξης (**iterative and incremental development process.)**
    - η ανάλυση, ο σχεδιασμός και η ανάπτυξη συμβαίνουν σε επαναλαμβανόμενους κύκλους και σε μικρότερα τμήματα κάθε φορά.

ITERATIVE AND INCREMENTAL DEVELOPMENT – ΕΠΑΝΑΛΗΠΤΙΚΗ/ΑΥΞΗΤΙΚΗ ΑΝΑΠΤΥΞΗ



OBJECT ORIENTED MODELLING

- **Identifying objects (Αναγνώριση αντικειμένων):**
  χρησιμοποιώντας encapsulation concepts, CRC cards, etc.

- **Organising the objects (Οργάνωση των αντικειμένων):**
  ταξινόμηση των αντικειμένων που έχουν αναγνωριστεί, έτσι ώστε παρόμοια αντικείμενα να μπορούν μετά να οριστούν στην ίδια κλάση (class)

- **Identifying relationships between objects (Προσδιορισμός σχέσεων μεταξύ αντικειμένων):**
  πως τα αντικείμενα επικοινωνούν μεταξύ τους και τα μηνύματα που ανταλλάζουν

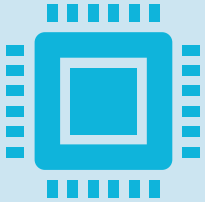
- **Defining behaviours (operations) of objects (Καθορισμός των συμπεριφορών (μεθόδων των αντικειμένων):**
  ο τρόπος με τον οποίο τα δεδομένα επεξεργάζονται εντός ενός αντικειμένου

- **Defining objects internally (Εσωτερικός ορισμός αντικειμένων):**
  Τι πληροφορίες, δεδομένα (data/attributes) αποθηκεύονται εντός των αντικειμένων

ΑΛΛΑ ΤΙ ΕΙΝΑΙ ΈΝΑ ΑΝΤΙΚΕΙΜΕΝΟ - OBJECT?

Η αντικειμενοστρεφής προσέγγιση αποσκοπεί στο να φέρει τον προγραμματισμό ενός συστήματος πιο κοντά στο να σκεφτόμαστε τον πραγματικό κόσμο. (make thinking about programming closer to thinking about the real world)

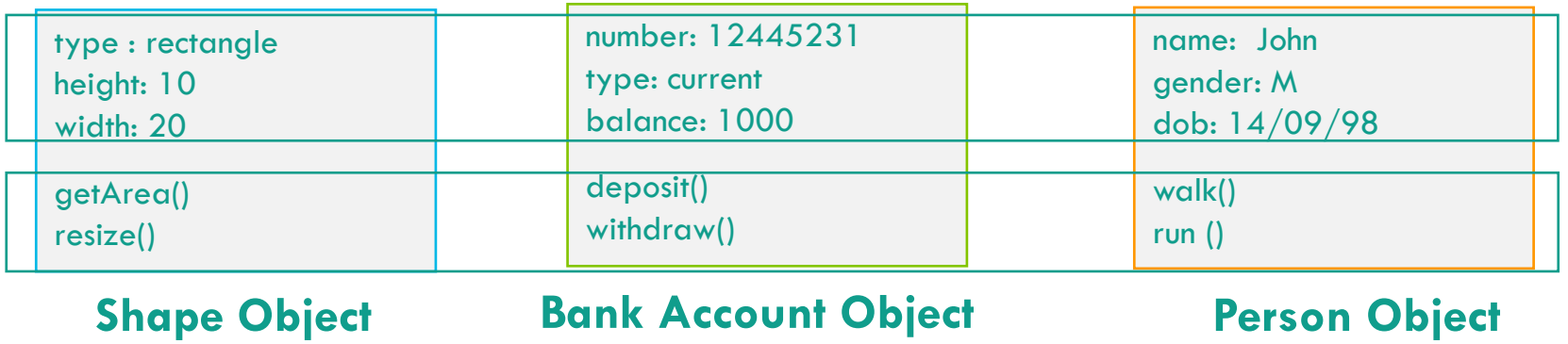
Τι είναι ένα αντικείμενο στον προγραμματισμό πρέπει πρώτα να αναρωτηθούμε τι είναι ένα αντικείμενο στον πραγματικό κόσμο.



Object: Τα αντικείμενα είναι οντότητες ενός συστήματος λογισμικού οι οποίες αντιπροσωπεύουν στιγμιότυπα πραγματικών οντοτήτων και οντοτήτων του συστήματος. Κάθε αντικείμενο χαρακτηρίζεται από ένα σύνολο ιδιοτήτων (attributes) και συμπεριφοράς (methods)

OBJECTS

- **Οντότητες που κάνουν encapsulate data and behaviour**
  - Στα αντικείμενα οι χρήστες κάνουν store data and associate behaviour
- **Τα δεδομένα (Data) είναι τα χαρακτηριστικά ενός αντικειμένου**
  - ιδιότητες που τα περιγράφουν (π.χ. ένα δοχείο μπορεί να είναι γεμάτο ή άδειο, μια λάμπα μπορεί να είναι αναμμένη ή σβηστή, ένα μήλο μπορεί να είναι κίτρινο ή κόκκινο
  - αυτά είναι τα **χαρακτηριστικά** κάθε αντικειμένου, πράγματα όπως το **χρώμα, το μέγεθος και το βάρος** - περιγράφουν την **τρέχουσα κατάσταση (state)** ενός αντικειμένου - η κατάσταση (state) ενός αντικειμένου είναι ανεξάρτητη από την κατάσταση του άλλου



OBJECTS

- **Τα αντικείμενα δεν είναι πάντα φυσικά αντικείμενα ή ορατά αντικείμενα, π.χ. αυτοκίνητο, σπίτι, προϊόν, μήλο.**
  - A date, a bank account, timer can be an object – they still meet our idea of an object

- **Κάθε αντικείμενο έχει ταυτότητα (identity)**
  - it can be referred to individually - μπορεί να αναφέρεται ξεχωριστά
  - it is distinguishable from other objects - διακρίνεται από άλλα αντικείμενα

- **Μπόρείτε να βάλετε την λέξη "το" "the" μπροστά τους?**
  - The mug, the apple, the bank account, the date.
  - Δεν θα λέγαμε the saving, the printing, αυτά είναι ρήματα (verbs) και δείχνουν συμπεριφορά (behaviours) των αντικειμένων
  - Τα **αντικείμενα** συνήθως αναπαραστούνται από **ουσιαστικά** (nouns) ενώ η συμπεριφορά τους (**operations/methods)** του αντικειμένου από **ρήματα** (verbs)

CLASSES - ΚΛΑΣΕΙΣ

- **Τα αντικείμενα και οι κλάσεις (classes) πάνε χέρι-χέρι.**
  - Δεν μπορούμε να μιλήσουμε για το ένα χωρίς να μιλήσουμε για το άλλο.
  - Το OOAD αφορά τις κλάσεις επειδή χρησιμοποιούμε κλάσεις για να δημιουργήσουμε αντικείμενα

- Μια **κλάση** είναι ένα σχέδιο ή ένα πρότυπο ή ένα σύνολο οδηγιών που χρησιμοποιούνται για τη δημιουργία ενός συγκεκριμένου τύπου αντικειμένου (object instances) - **περιγράφει τι θα είναι ένα αντικείμενο, αλλά δεν είναι το ίδιο το αντικείμενο.**

- Μια κλάση έρχεται πάντα πρώτη

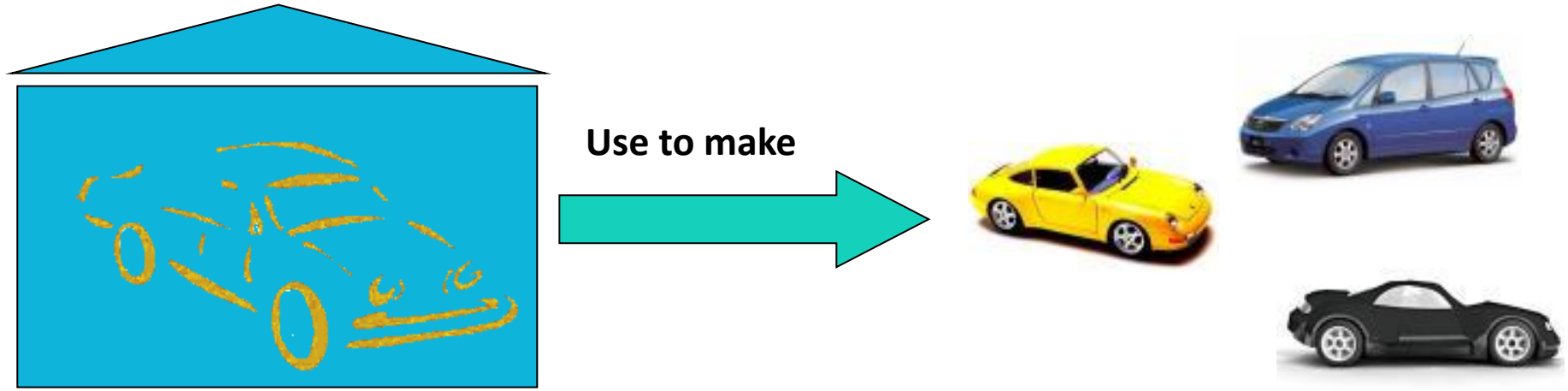
- Μια αφαιρετικότητα δεδομένων (abstraction) η οποία καθορίζει τα attributes /behaviour ενός συνόλου αντικειμένων
  - Κάθε κλάση, περιγράφει ένα σύνολο αντικειμένων με: ***κοινά χαρακτηριστικά*** (attributes), ***κοινή συμπεριφορά*** (operations/methods), ***κοινές σχέσεις*** με άλλα αντικείμενα και ***κοινή σημασιολογία***
  - ***Όλα τα αντικείμενα μιας κλάσης είναι πανομοιότυπα ως προς τη δομή (χαρακτηριστικά) και τη συμπεριφορά, αλλά περιέχουν διαφορετικά δεδομένα στα χαρακτηριστικά τους.***

THE DIFFERENCE BETWEEN OBJECTS AND CLASSES

- Μια κλάση είναι ένα σχέδιο, ή πρωτότυπο, που ορίζει τα χαρακτηριστικά και τις μεθόδους που είναι κοινές για όλα τα αντικείμενα ενός συγκεκριμένου είδους.

Class car:

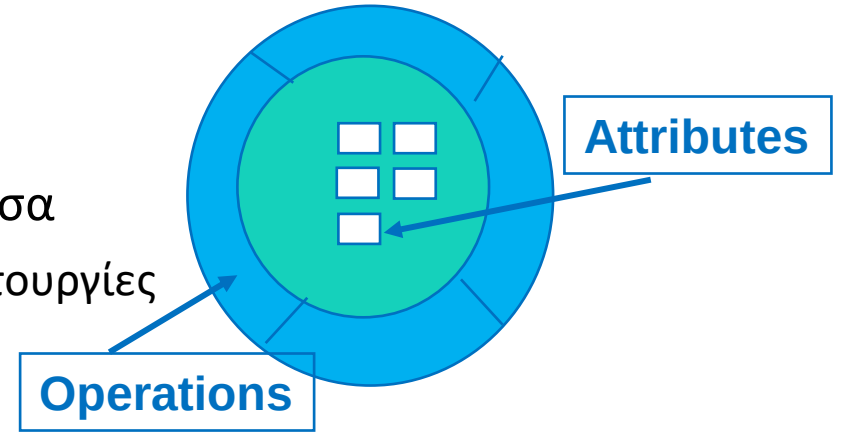
- Attributes: make, colour, type, weight etc.
- Behaviour: start_engine(); drive()



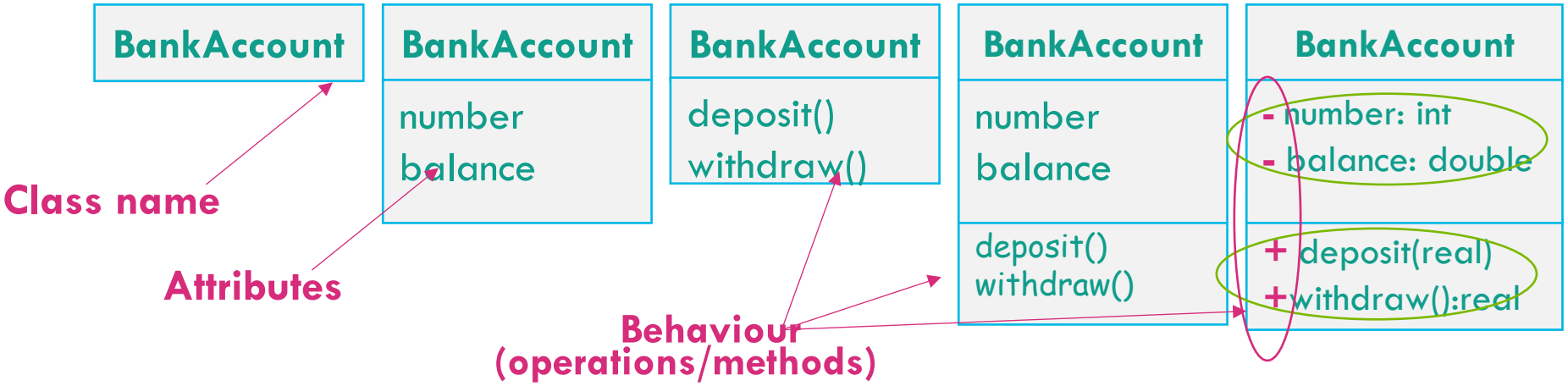
Class (the blueprint)

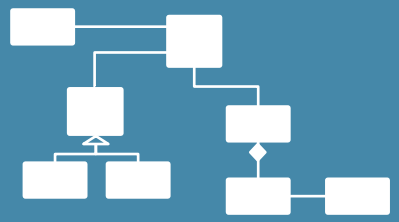
Objects

UML CLASS NOTATION



- Απλά αναπαρίσταται ως ένα κουτί με το όνομα της κλάσης μέσα
  - το οποίο δείχνει επίσης τα χαρακτηριστικά (attributes) και τις λειτουργίες (attributes and operations)
  - the complete signature of an attribute /operation is:
    - <attributeName>:<type>
    - <operationName(parameterName:paramType…)>:<returnType>
    - **Visibility**: private (**-**), public (**+**), protected (**#**), or package(**~**)

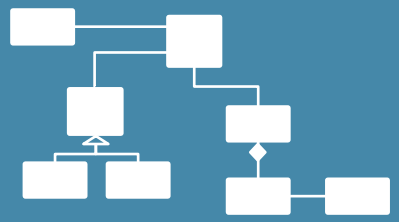




ATTRIBUTE TYPE
CLASS/OBJECTS

- Συνήθεις Τύποι Attribute και Τι Σημαίνουν:

Τύπος	Περιγραφή	Παράδειγμα
int (integer)	Ακέραιος αριθμός (χωρίς δεκαδικά)	ηλικία = 25
double ή real	Δεκαδικός αριθμός (με ακρίβεια)	βάρος = 68.5
string	Κείμενο ή λέξεις	όνομα = "Μαρία"
boolean	Αληθές ή ψευδές (True/False)	είναιΕνεργό = true
char	Ένας μόνο χαρακτήρας	φύλο = 'Μ'
date	Ημερομηνία	ημερομηνίαΓέννησης = 14/09/1998
float	Δεκαδικός αριθμός με λιγότερη ακρίβεια από το double	ποσοστό = 75.2



UML NOTATION
CLASS VS OBJECTS

Class

Book
title: String author : String isbn: String publisher: String price: Real
getQuantity() get_bookDetails() set_bookDetails() deletebook()

Object – Instance of Book

Book : myBook
title = “Software Engineering” author = “Pressman” isbn = “2-5345-53455-6” publisher = “McGrawHill” price = 40.00
getQuantity() get_bookDetails() set_bookDetails() deletebook()

OBJECT-ORIENTED OBJECT/CLASS PRINCIPLES

Identity - ένα αντικείμενο γνωρίζει ποιο είναι

Classification - ένα αντικείμενο γνωρίζει την κλάση (class) στην οποία ανήκει
classify objects to later create similar objects as needed

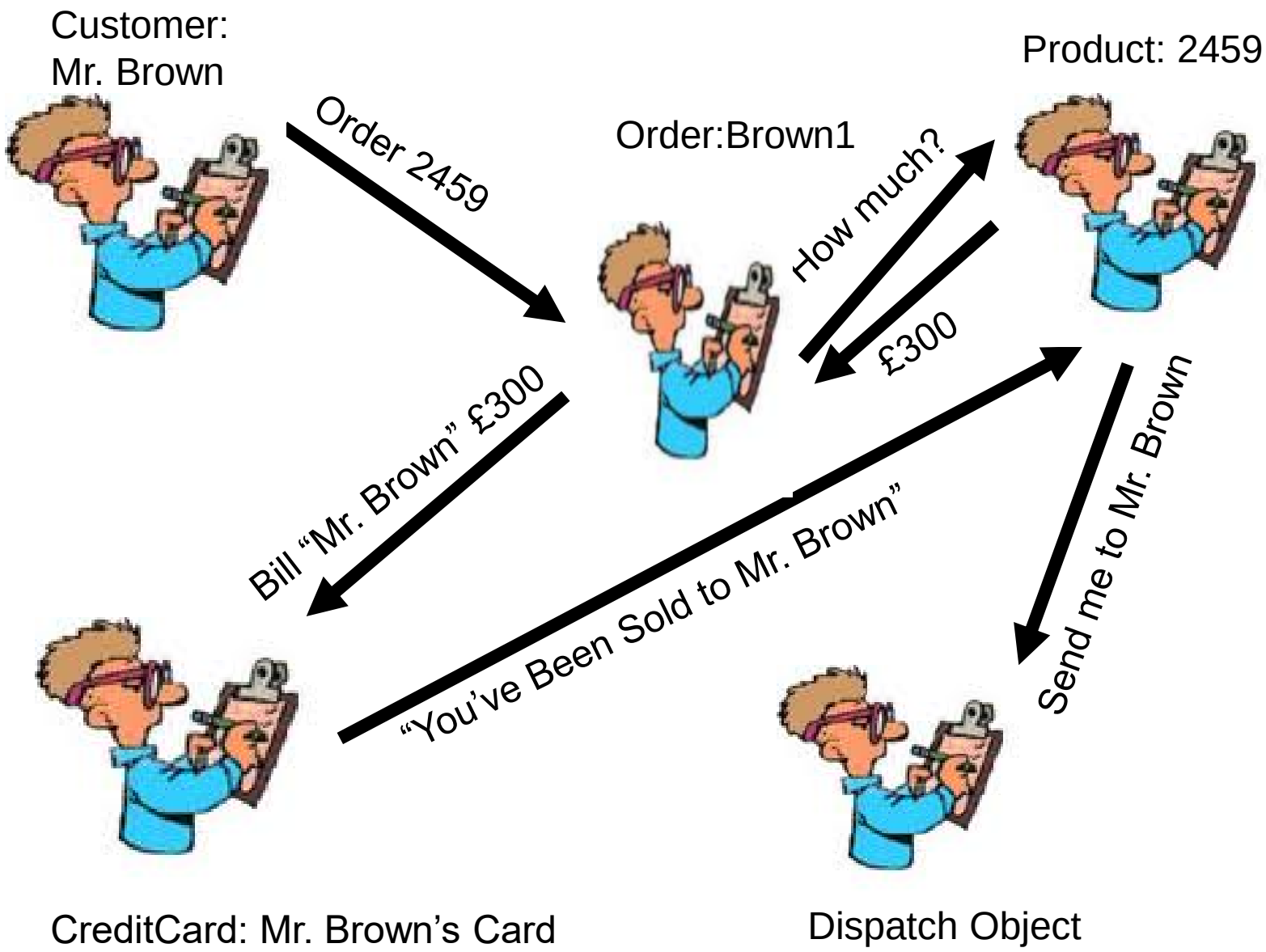
Relationships - ένα αντικείμενο ξέρει με ποια αλλά αντικείμενα πρέπει να αλληλοεπιδρά
this helps to determine the inputs and outputs of an object

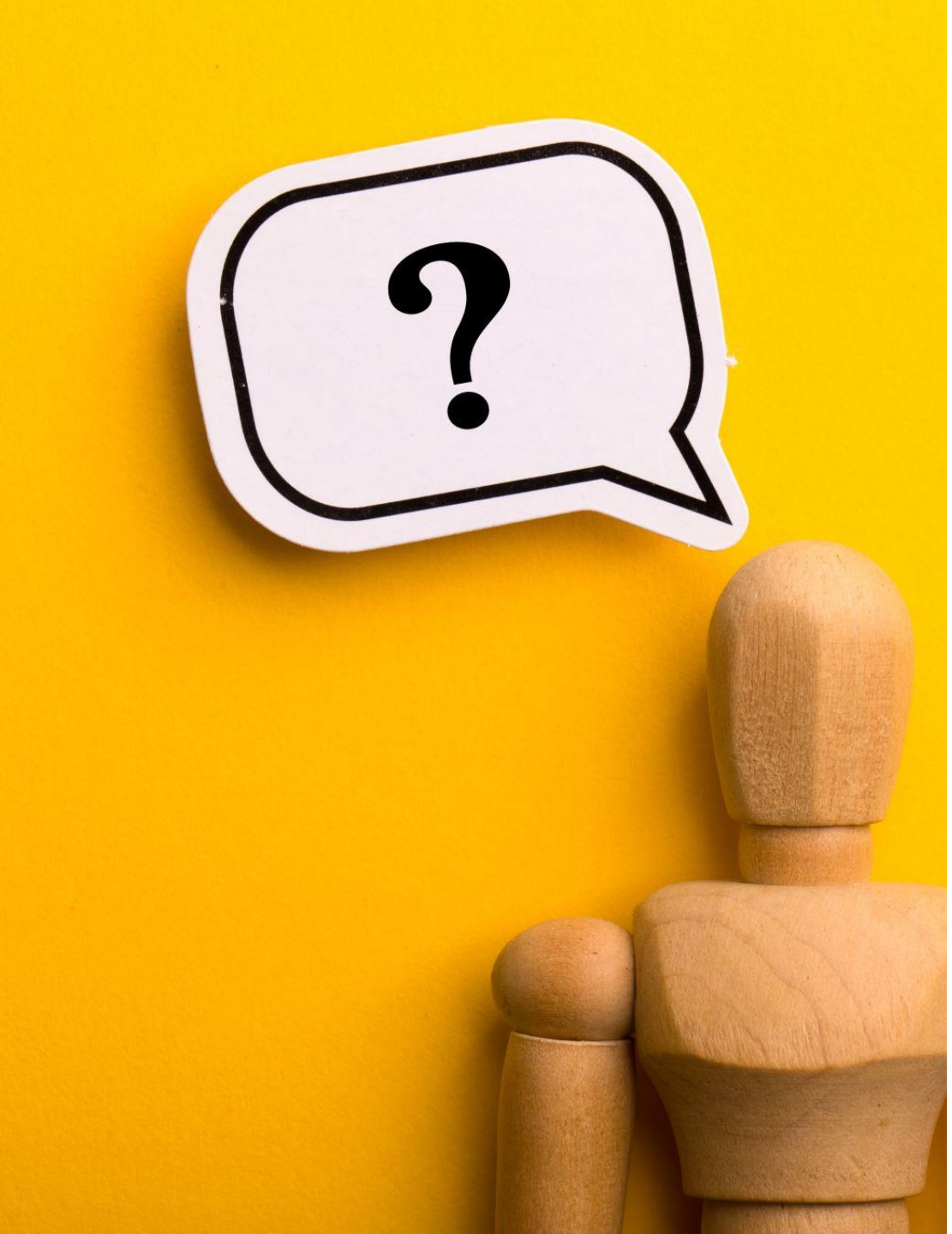
Abstract behaviour - τα αντικείμενα εκτελούν ενέργειες/λειτουργίες, έχουν συμπεριφορά

the way data is processed within an object

Για να ορίσετε τα αντικείμενα, κάντε ερωτήσεις όπως

- Ποια είναι τα πράγματα (things) (classes/objects) τα οποία εμπλέκονται στο σύστημα?
- Τι πρέπει να γνωρίζω για να είμαι σε θέση να εκτελέσω το ρόλο μου; ? Τι δεδομένα (data) πρέπει να κατέχω ? **(attributes)**
- Τι πρέπει να κάνω για να μπορώ να εκτελώ το ρόλο μου; Τι λειτουργίες πρέπει να μπορώ να εκτελώ? Ποιο είναι το behaviour μου? **(methods)**
- Ποιον πρέπει να γνωρίζω για να μπορώ να εκτελώ το ρόλο μου; Με ποιες άλλες κλάσεις πρέπει να συνδέομαι/συνεργάζομαι? **(relations/visibility)**


Customer:
Mr. Brown
Order 2459
Order:Brown1
How much?
Product: 2459
£300
Bill "Mr. Brown" £300
"You've Been Sold to Mr. Brown"
Send me to Mr. Brown
CreditCard: Mr. Brown's Card
Dispatch Object



BREAKOUT SESSION

ΑΝΑΓΝΩΡΙΣΗ ΚΛΑΣΕΩΝ/ΧΑΡΑΚΤΗΡΙΣΤΙΚΩΝ ΚΑΙ ΜΕΘΟΔΩΝ

Σενάριο: Φανταστείτε ότι σχεδιάζουμε ένα πληροφοριακό σύστημα για μια **δανειστική βιβλιοθήκη**. Το σύστημα επιτρέπει στους χρήστες να δανείζονται και να επιστρέφουν βιβλία, να βλέπουν τη διαθεσιμότητα τίτλων και να κάνουν κράτηση βιβλίων.

Ερωτήσεις:

1.Ποιες είναι οι βασικές κλάσεις που θα μπορούσαμε να εντοπίσουμε σε αυτό το σύστημα;

2.Τι χαρακτηριστικά (ιδιότητες) έχει κάθε κλάση; (π.χ. όνομα, κωδικός, email, τίτλος βιβλίου, ISBN)

3.Ποιες βασικές συμπεριφορές ή λειτουργίες (methods) θα μπορούσε να έχει κάθε κλάση; (π.χ. δανεισμός, επιστροφή, αναζήτηση)

APIE OBJECT ORIENTED PRINCIPLES

Υπάρχουν διάφορες έννοιες στην αντικειμενοστρεφή προσέγγιση που μας βοηθούν να αντιμετωπίσουμε τα προβλήματα.

Το OOAD βασίζεται σε ένα σύνολο αρχών που χρησιμεύουν στη διαχείριση και τη μείωση της πολυπλοκότητας του συστήματος.

APIE - Abstraction, Polymorphism, Inheritance, Encapsulation

ABSTRACTION - ΑΦΑΙΡΕΤΙΚΟΤΗΤΑ

Abstraction - Αφαιρετικότητα: αποτύπωση βασικών πτυχών, αγνόηση άσχετων λεπτομερειών

- Περιλάβετε μόνο τα σχετικά και σημαντικά στοιχεία σε μια κλάση
- Υποστηρίζεται από δυο κύριες ιδέες:
  - **Information Hiding (απόκρυψη πληροφοριών)**
  - **Encapsulation (ενθυλάκωση)**

INFORMATION HIDING - ΑΠΟΚΡΥΨΗ ΠΛΗΡΟΦΟΡΙΩΝ

Information Hiding

- **Μια κλάση θα πρέπει να αποκρύπτει από άλλες κλάσεις τις λεπτομέρειες του τρόπου με τον οποίο υλοποιείται εσωτερικά ως κώδικας υπολογιστή.**

- Επιτρέπει τα δεδομένα (attributes) να μην είναι ορατά στον έξω κόσμο, **έτσι οι μέθοδοι (methods) είναι αυτές που επιτρέπουν σε άλλα αντικείμενα να έχουν πρόσβαση στα δεδομένα με ελεγχόμενο τρόπο.**

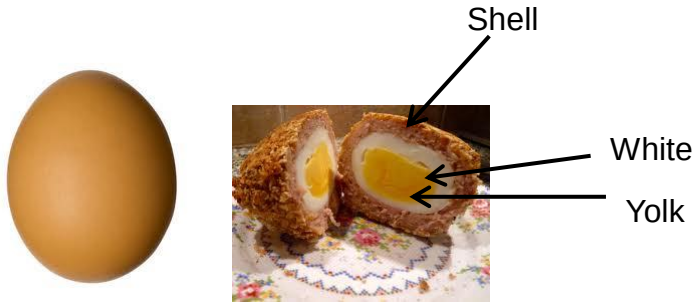
- Η απόκρυψη πληροφοριών είναι χρήσιμη καθώς οι κλάσεις και τα δεδομένα τους **δεν μπορούν να αλλοιωθούν από άλλες κλάσεις που δεν πρέπει να τις χρησιμοποιούν**.

Clipboard example: δεν μπορούμε να δούμε τις πληροφορίες στο σημειωματάριο (clipboard) του άλλου. Πρέπει να ζητήσουμε άδεια για να έχουμε πρόσβαση σε αυτές τις πληροφορίες.

ENCAPSULATION - ΕΝΘΥΛΑΚΩΣΗ

Encapsulation:

- **Συνδυασμός τόσο των χαρακτηριστικών (attributes) όσο και της συμπεριφοράς (methods) σε μια κλάση και απόκρυψη της μη σχετικής λεπτομέρειας σε μια δεδομένη κλίμακα**
  - Εάν βάλεις τα δεδομένα και την συμπεριφορά πίσω μαζί σε ένα αντικείμενο τότε υποστηρίζεται η αφαιρετικότητα (abstraction) επειδή τα αντικείμενα μπορούν να χρησιμοποιηθούν σαν **building blocks** που μπορούν να συναρμολογηθούν για να δημιουργήσουν το σύστημα.
  - **Encapsulation allows for a more robust model, easier to reuse**



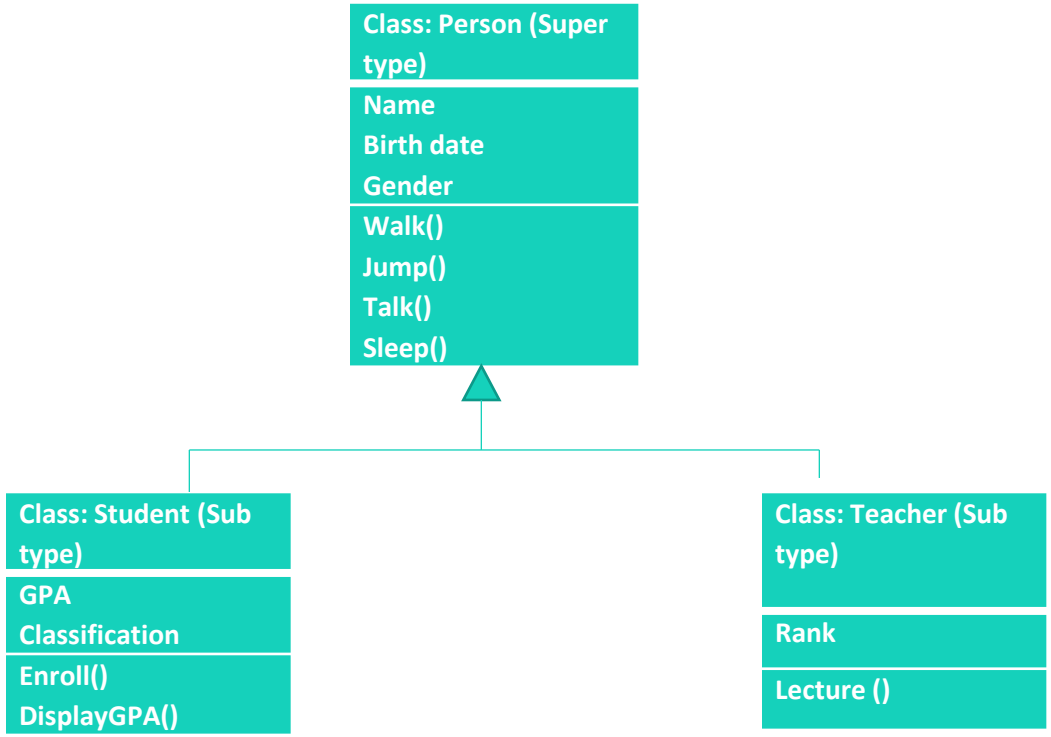
Example: Object is like an Egg

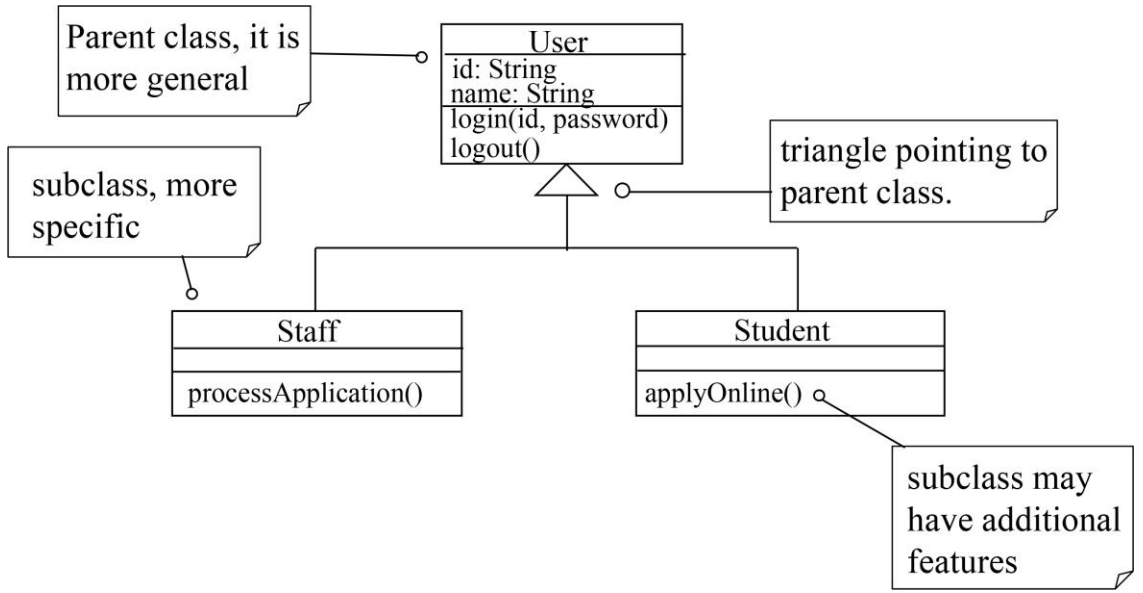
- Ο κρόκος είναι τα δεδομένα (**attributes**) που περιβάλλονται από το ασπράδι. Το ασπράδι είναι οι μέθοδοι (methods) που ενεργούν στα δεδομένα..
- Το κέλυφος περιβάλλει όλο το αυγό και το κρατάει ενωμένο ενώ κρύβει το εσωτερικό του από τον εξωτερικό κόσμο. Το κέλυφος είναι η διεπαφή και το μόνο πράγμα που φαίνεται.
- **Με την ενθυλάκωση και την απόκρυψη πληροφοριών, η επιβάρυνση κατά τη συντήρηση μειώνεται, καθώς είστε σίγουροι ότι όταν διορθώνετε ένα πρόβλημα σε μια κλάση, επηρεάζεται μόνο ο κώδικας της συγκεκριμένης κλάσης.**

INHERITANCE - ΚΛΗΡΟΝΟΜΙΚΟΤΗΤΑ

Inheritance

- Ένα αντικείμενο από μια πιο συγκεκριμένη κλάση μπορεί να κληρονομήσει τα χαρακτηριστικά(attributes) και τις μεθόδους (methods) από μια πιο γενική κλάση.



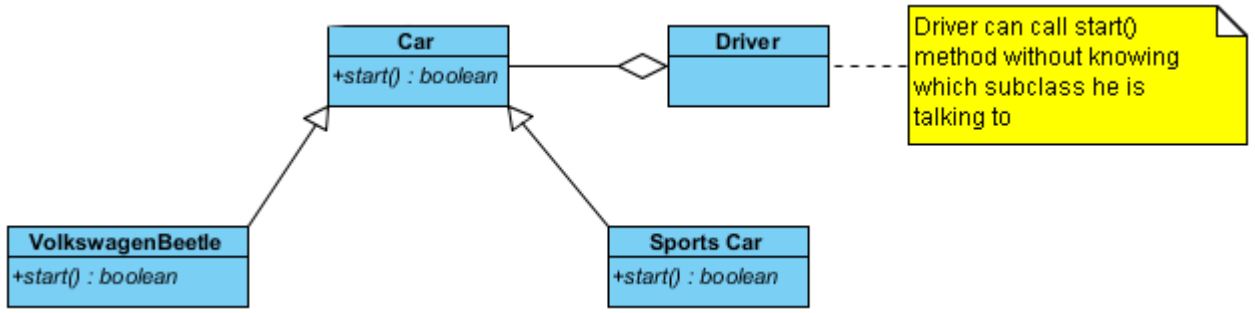


POLYMORPHISM - ΠΟΛΥΜΟΡΦΙΣΜΟΣ

Polymorphism

- Τα γενικά χαρακτηριστικά μιας κλάσης μπορούν να κληρονομηθούν αλλά να αντικατασταθούν αν χρειαστεί.
- Επιτρέπει σε μεθόδους διαφορετικών κλάσεων να έχουν το ίδιο όνομα, ενώ οι λεπτομέρειες της πραγματικής λειτουργίας που εκτελείται μπορεί να είναι διαφορετικές.

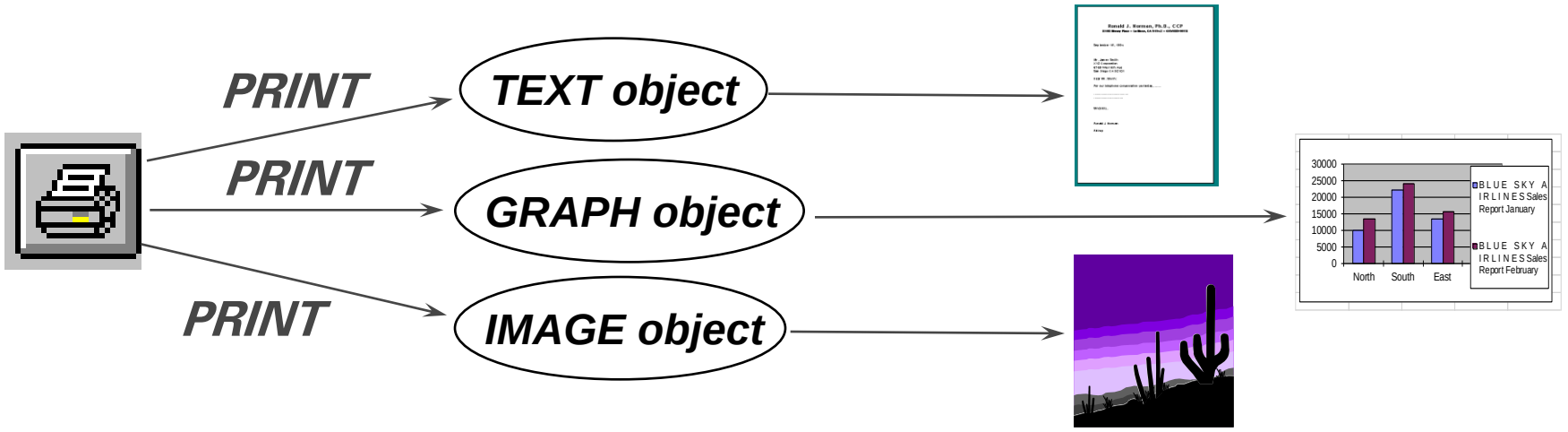
E.g. "Close method" a door can "swing shut" a window can "slide downwards"



Πώς σχετίζεται ο πολυμορφισμός με την αποστολή μηνυμάτων μεταξύ αντικειμένων;

Το αιτούμενο αντικείμενο γνωρίζει ποια συμπεριφορά (method) να ζητήσει και από ποιο αντικείμενο, αλλά δεν χρειάζεται να ανησυχεί για τον τρόπο με τον οποίο επιτυγχάνεται αυτή η συμπεριφορά.

POLYMORPHISM EXAMPLE



- **Η κληρονομικότητα και ο πολυμορφισμός μπορούν να βελτιώσουν την επαναχρησιμοποίηση.**
- **Αυτό μειώνει το κόστος σχεδιασμού, προγραμματισμού και testing.**

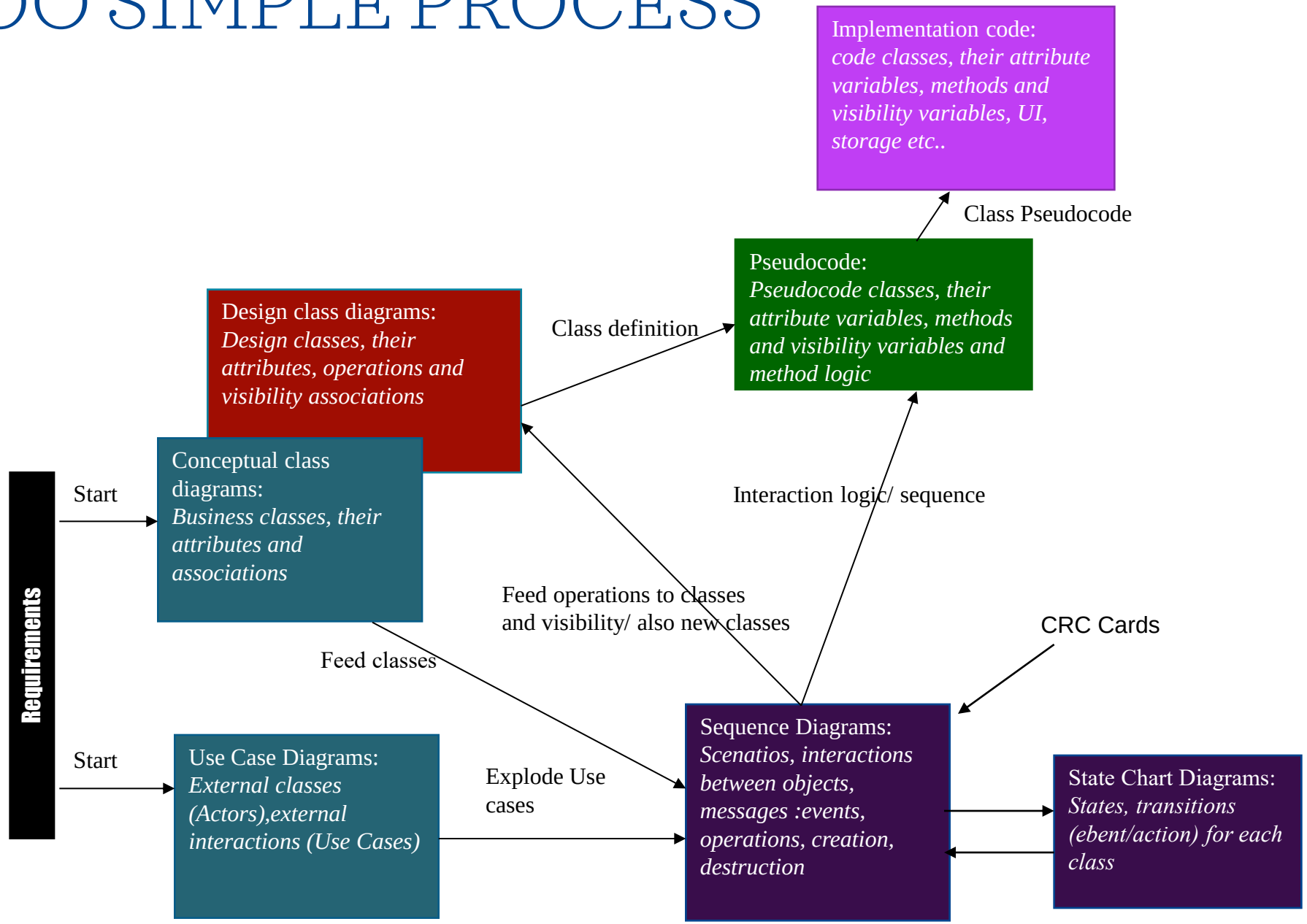
ΠΟΙΟ ΕΙΝΑΙ ΤΟ ΑΠΟΤΕΛΕΣΜΑ ΤΗΣ ΑΝΤΙΚΕΙΜΕΝΟΣΤΡΕΦΟΥΣ ΑΝΑΛΥΣΗΣ

Το αποτέλεσμα της αντικειμενοστρεφούς ανάλυσης του συστήματος είναι διάφορα μοντέλα.

A model should:

- να χρησιμοποιεί μια τυποποιημένη γλώσσα μοντελοποίησης (θα χρησιμοποιήσουμε τη UML)
- να είναι κατανοητό από τους πελάτες και τους χρήστες
- να οδηγεί τους μηχανικούς λογισμικού σε γνώσεις σχετικά με το σύστημα
- να παρέχει αφαίρετικοτητα

AN OO SIMPLE PROCESS



DIS501 Ανάλυση και Σχεδίαση Πληροφοριακών Συστημάτων

Lecture 6: Use Case Modelling: Διαγράμματα Περιπτώσεων χρήσης με UML

ΒΥ: ΔΡ ΑΥΓΟΥΣΤΑ ΚΥΡΙΑΚΙΔΟΥ ΖΑΧΑΡΟΥΔΙΟΥ

UML USE-CASE MODELLING

- Μια προσέγγιση της ανάλυσης συστημάτων με επίκεντρο τον χρήστη (**user-centred**)
- Αναλύει τις **απαιτήσεις** διερευνώντας τις **αλληλεπιδράσεις των χρηστών** με το σύστημα.
- Προσδιορίζει και περιγράφει τις λειτουργίες του συστήματος από τη σκοπιά των **εξωτερικών χρηστών**
- Οι περιπτώσεις χρήσης (Use cases) προκύπτουν από τις απαιτήσεις και ικανοποιούν τις απαιτήσεις.
- **Μοντελοποιεί τις εμφανείς *λειτουργικές απαιτήσεις (Functional Requirements)***
  - "System will print receipt when payment is processed"
  - "Student needs to register in order to view their grades"
  - "Customer can search for books"
  - "Customer can write reviews"

USE CASE MODELLING: **ΔΙΑΔΙΚΑΣΙΑ**

Η διαδικασία περιλαμβάνει την τεκμηρίωση

Who/what **initiates** an interaction with the system (ποιος/τι αλληλοεπιδρά με το σύστημα)

What **information goes into** the system (τι πληροφορίες εισέρχονται στο σύστημα)

What information **comes out** and (τι πληροφ… …προκύπτουν από το σύστημα)

What the **m**… …e. what they get out of it – ***the f***… …λειτουργία που του παρέχεται)

Use-Case	What is the functional goal?
Actor	Who wants it?
Scenario	What is needed to accomplish goal? – *business tasks*.

USE CASE MODELLING: ΔΙΑΔΙΚΑΣΙΑ

- Προσδιορίζουμε τους **Actors**
- Προσδιορίζουμε τα **Use-Cases**
- Προσδιορίζουμε το **System Boundaries**
- Παρουσιάζουμε τις περιπτώσεις χρήσης (use-cases) και τις σχέσεις μεταηύ αυτών και των χρηστών (actors) με την δημιουργία ενώς διαγράμματος περιπτώσεων χρήσης (**Use-Case Diagram)**
- Δημιουργούμε ένα **Use-Case scenario** για κάθε περίπτωση χρήσης(**use-case**)
- Βελτιώνουμε το μοτέλο σε συνεργασία με τον πελάτη

UML ACTORS

Αντιπροσωπεύουν **οποιονδήποτε/οτιδήποτε (anyone/anything)** το οποίο είναι εκτός του συστήματος και χρειάζεται να **αλληλοεπιδράσει με αυτό.**

- Οι πιο εμφανείς actors είναι οι **άνθρωποι που θα χρησιμοποιήσουν** το σύστημα
- **Άλλα συστήματα** (databases, servers maintained by other people, legacy systems, bank systems)
  - Τα οποία πρέπει να αλληλεπιδράσουν με το σύστημα μας - need to interact with our system

Initiate system activity, e.g a **use-case**, for the purpose of completing some system functionality

Ο actor αντιπροσωπεύει ένα ρόλο τον οποίο παίζει ο χρήστης όταν αλληλοεπιδρά με το σύστημα.

eg. DreamVideo case study

- Sales Staff
- Customer

Roles, not individual users or people

- Smith 
- SalesStaff 

ΓΙΑ ΝΑ ΠΡΟΣΔΙΟΡΙΣΕΤΕ ΤΟΥΣ ACTORS: ΚΑΝΕΤΕ ΑΥΤΕΣ ΤΙΣ ΕΡΩΤΗΣΕΙΣ

Ποιος θα χρησιμοποιήσει το σύστημα?

Ποιος θα εισάγει, θα χρησιμοποιεί, θα ενημερώνει ή θα διαγράφει πληροφορίες από το σύστημα;

Ποιος ενδιαφέρεται για μια συγκεκριμένη λειτουργία ή υπηρεσία που παρέχεται από το σύστημα;

Ποιος θα υποστηρίζει και θα συντηρεί το σύστημα;

Ποιοι είναι οι εξωτερικοί πόροι του συστήματος;

Ποια άλλα συστήματα θα πρέπει να αλληλοεπιδρούν με το υπό ανάπτυξη σύστημα για να μπορεί να ολοκληρώσει τις λειτουργίες του;

Review the list of stakeholders

Δεν είναι όλοι οι stakeholders actors. Αλλά αυτή είναι μια καλή αρχή για να αναγνωρίσουμε τους πιθανούς actors.

UML ACTORS: EXAMPLE

Online bookstore case study example Actors:

Customer - can order books online, browse catalogue, write reviews etc.

Sales Staff - can place orders on behalf of customers, write reviews, reply to enquiries etc.

Directors - can print sales reports

Warehouse System - needs to interact with the system to maintain audit, check availability

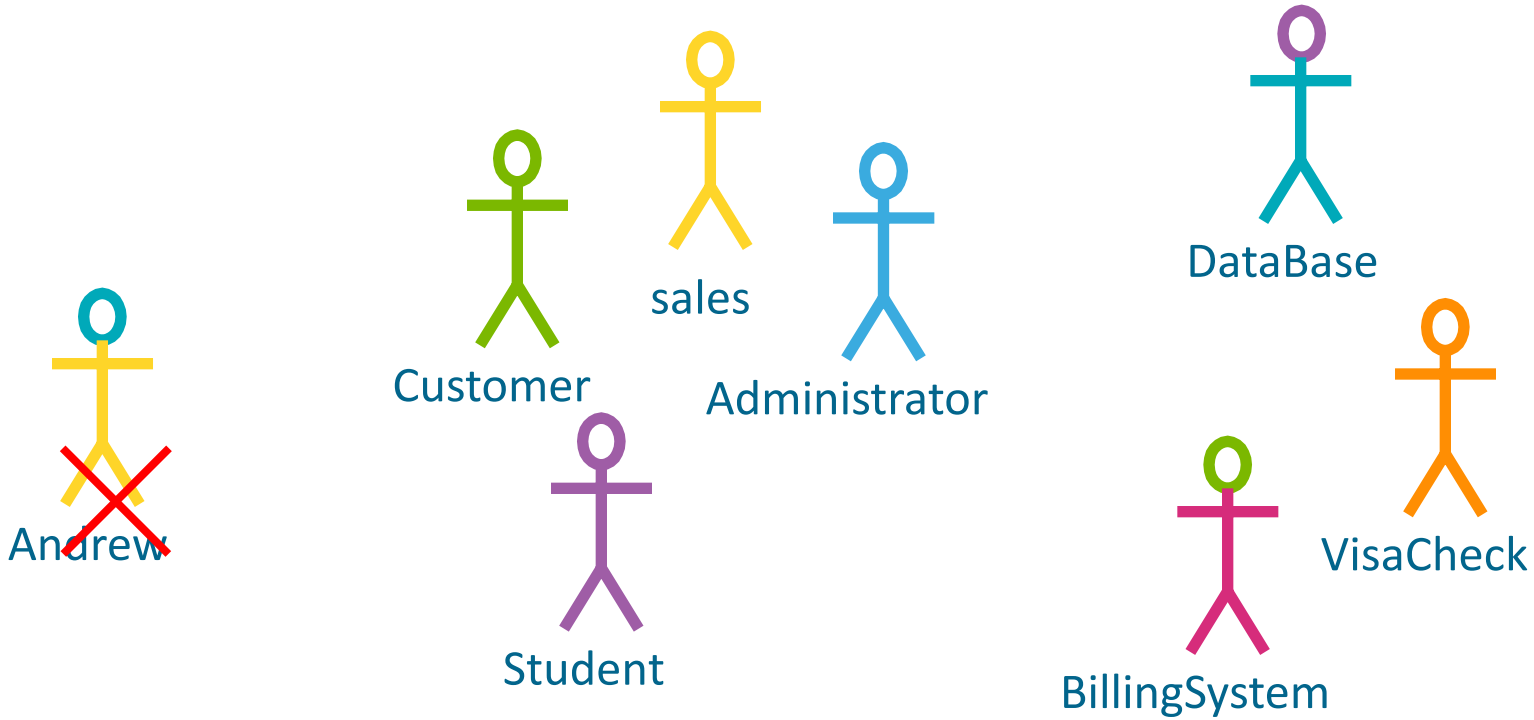
VisaCheck - needs to interact with system to verify credit card details

Human Actors

Να θυμάστε οι actors είναι οποιοσδήποτε η οτιδήποτε ενεργοποιεί μια λειτουργία του συστήματος ή πρέπει να αλληλεπιδράσει με το υπό ανάπτυξη σύστημα για την ανταλλαγή πληροφοριών

Machine Actors

UML ACTOR NOTATION



ΤΥΠΟΙ ACTORS
ΠΡΩΤΕΥΩΝ (PRIMARY) ACTORS

Primary Actors

- **Αλληλεπιδρούν άμεσα με το σύστημα**
- **Απαιτούν τη βοήθεια του συστήματος, και έτσι κάνουν Initiate ξεκινούν μια λειτουργία του συστήματος**
- Είναι αυτοί για τους οποίους σχεδιάζεται το μελλοντικό σύστημα
- Είναι αυτοί που εκπληρώνουν τους στόχους τους με τη χρήση του συστήματος
- E.g: **Borrower** in a Library System, **Student or Lecturer** in a University system
  Customer in an online shopping system.

ΤΥΠΟΙ ACTORS

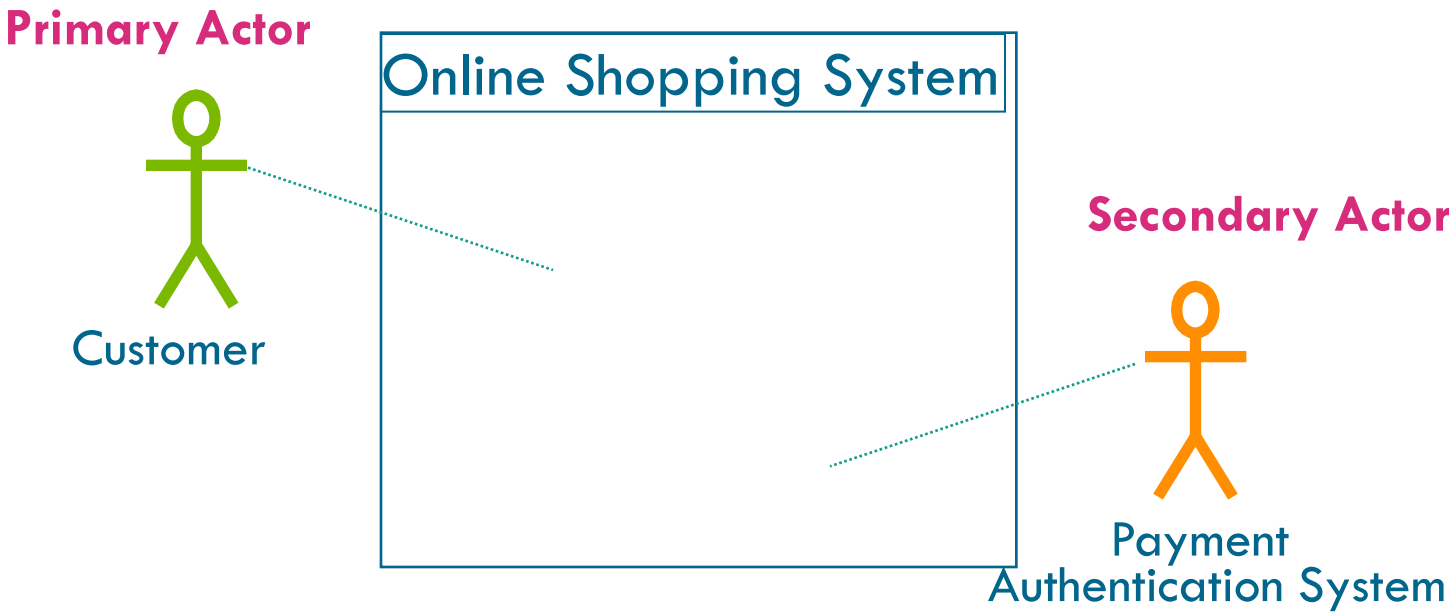
ΔΕΥΤΕΡΕΥΩΝ (SECONDARY) ACTORS

Secondary Actors

- **Αλληλεπιδρούν έμμεσα με το σύστημα**
- **Το σύστημα χρειάζεται την βοήθεια τους**
- Παρέχουν μια υπηρεσία στο μελλοντικό σύστημα, π.χ. μια ***βοηθητική υπηρεσία***
- Εκπληρώνουν έμμεσα τους στόχους του χρήστη παρέχοντας στο σύστημα βοηθητικές υπηρεσίες που θα ολοκληρώσουν αυτούς τους στόχους

E.g: **Database Administrator** in a Registry System,
VisaCheck system in a payment system,
Warehouse system in an e-commerce system

ΤΥΠΟΙ ACTORS: UML REPRESENTATION



Οι Actors τοποθετούνται εκτός του συστήματός. Όλοι οι actors σε ένα διάγραμμα πρέπει να σχετίζονται με τουλάχιστον μία περίπτωση χρήσης (use-case).

ΜΟΝΤΕΛΟ ΠΕΡΙΠΤΩΣΕΩΝ ΧΡΗΣΗΣ (USE CASE MODELLING): ΟΡΙΣΜΟΣ ΠΕΡΙΠΤΩΣΗΣ ΧΡΗΣΗΣ (USE CASE)

- **Οι λειτουργίες που υποστηρίζονται από το σύστημα.**
- Περιγράφουν τις λειτουργικές απαιτήσεις (**functional requirements) που πρέπει το σύστημα να υποστηρίζει**
  - E.g **Place Order** **or** **Write Review** **or** **Search for products**
- **Τεχνική βασισμένη σε σενάρια (Scenario-based) στη UML**
  - Κάθε περίπτωση χρήσης περιγράφεται ως μια σειρά βημάτων του τρόπου με τον οποίο ένα συγκεκριμένο σύστημα μπορεί να χρησιμοποιηθεί από τους actors (τελικούς χρήστες ή άλλα συστήματα)
- Μια περίπτωση χρήσης (use case) ενεργοποιείται από τους actors (συνήθως τους Primary Actors)
  - Ένας actor μπορεί να συσχετίζεται με περισσότερα από ένα use-cases

USE CASES: UML NOTATION

Όνομα use case: Συνδυασμός ρήματος-ουσιαστικού – για να υποδηλώνει μια ενέργεια, κάτι που κάνει το σύστημα.

- **Κάθε use case** αντιπροσωπεύει μια **λειτουργία** την οποία ενεργοποιεί ο actor. Είναι μια περίπτωση χρήσης του συστήματος

- **Use-Cases αντιμετωπίζουν το σύστημα ως μαύρο κουτί**

  - Καταγράφουν ***ποιος*** **(actor)** κάνει ***τι (πως αλληλεπιδρά με το σύστημα*** **(interaction)** , για ποιο ***σκοπό*** **(goal),**
    χωρίς να ασχολείται με τα εσωτερικά του συστήματος.

IDENTIFYING USE-CASES

Για κάθε **actor** αποφασίστε ποιες λειτουργίες (**functions**) χρειάζονται

University Online System Example:

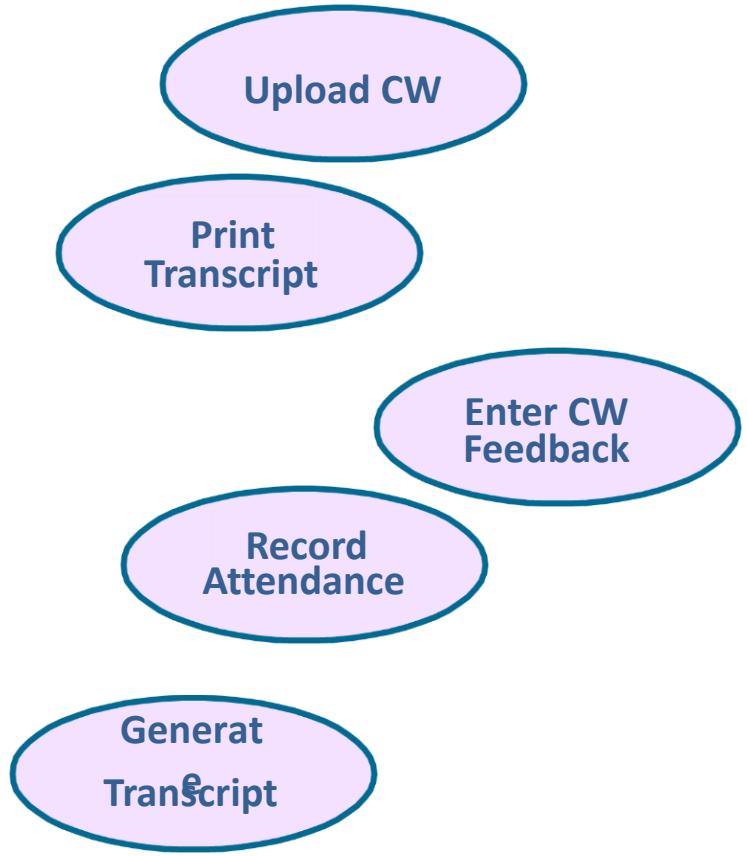
Student - upload coursework,

- print transcript

Lecturer - enter coursework feedback

- record attendance

StudentRecords - generate transcripts



IDENTIFYING USE-CASES

Χρήση των προδιαγραφών απαιτήσεων για τον προσδιορισμό των περιπτώσεων χρήσης από την οπτική γωνία ενός actor

- Ποιες είναι οι κύριες λειτουργίες (λειτουργικότητα που απαιτείται) για κάθε actor; (λειτουργικές απαιτήσεις)(functional requirements)
- Θα πρέπει ο actor να διαβάσει/γράψει/αλλάξει οποιαδήποτε πληροφορία του συστήματος;
- Θα πρέπει ο actor να ενημερώνει το σύστημα για εξωτερικές αλλαγές;
- Επιθυμεί ο actor να ενημερώνεται για απρόβλεπτες αλλαγές;

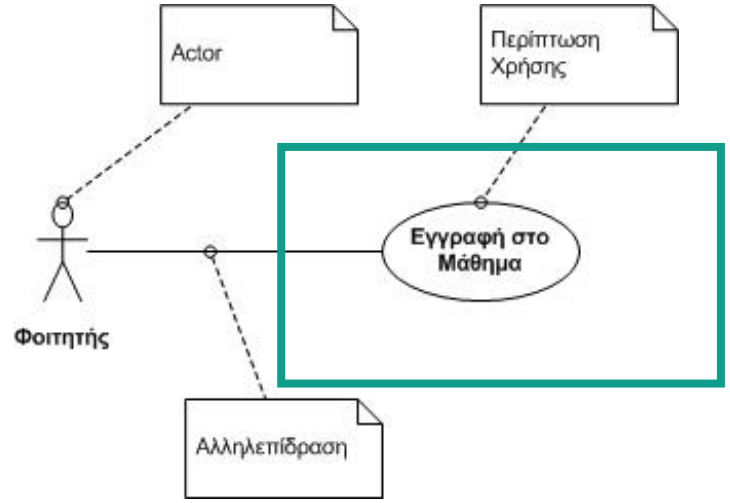
Θυμηθείτε ότι οι περιπτώσεις χρήσης αποτελούν το σημείο εκκίνησης για το δυναμικό μοντέλο (dynamic model).

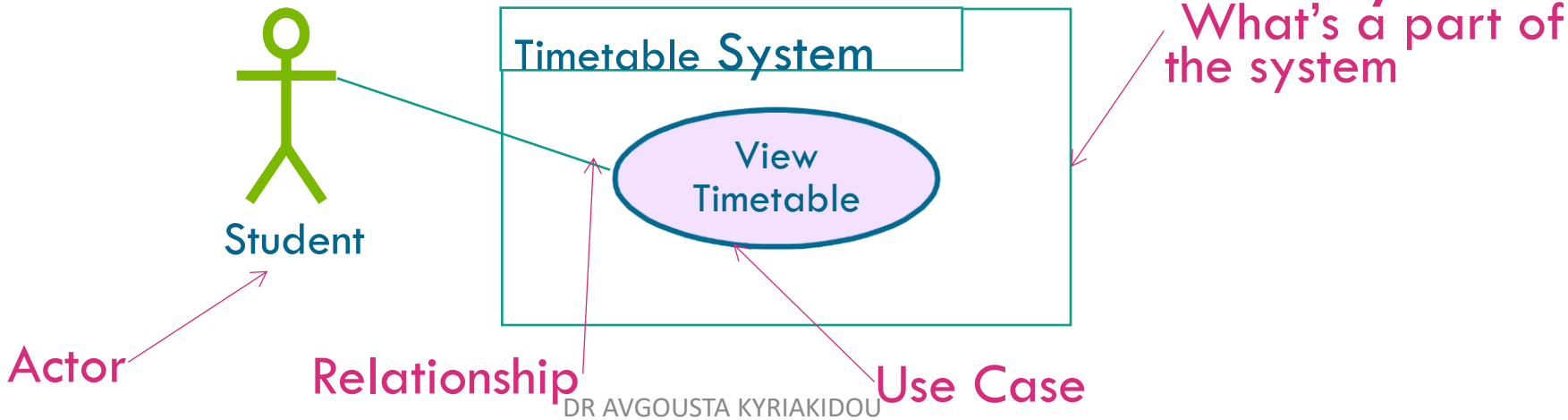
- Απεικονίζουν λειτουργία
- Αλλά δεν προσδιορίζουν τη συμπεριφορά ενός συστήματος η να δείχνουν την σειρά με την οποία συγκεκριμένες λειτουργίες ενεργοποιούνται. (Αυτό είναι κάτι που προσδιορίζεται στο object sequence model)
- Τα διαγράμματα περιπτώσεων χρήσης δεν δείχνουν πώς ανταλλάσσονται τα μηνύματα και δεν είναι διαγράμματα ροής

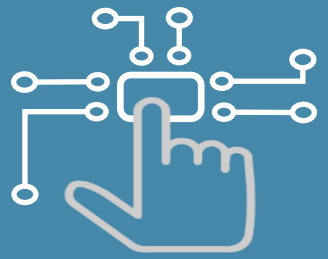
USE CASE DIAGRAMS

- **Τέσσερα στοιχεία:**
  - System – or part of system (καθορισμένα όρια)
  - Actors
  - Use-cases
  - Relationships

Ένα σύστημα μπορεί να απαιτεί πολλά διαγράμματα περιπτώσεων χρήσης και είναι σκόπιμο να διατηρούνται απλά και άμεσα.

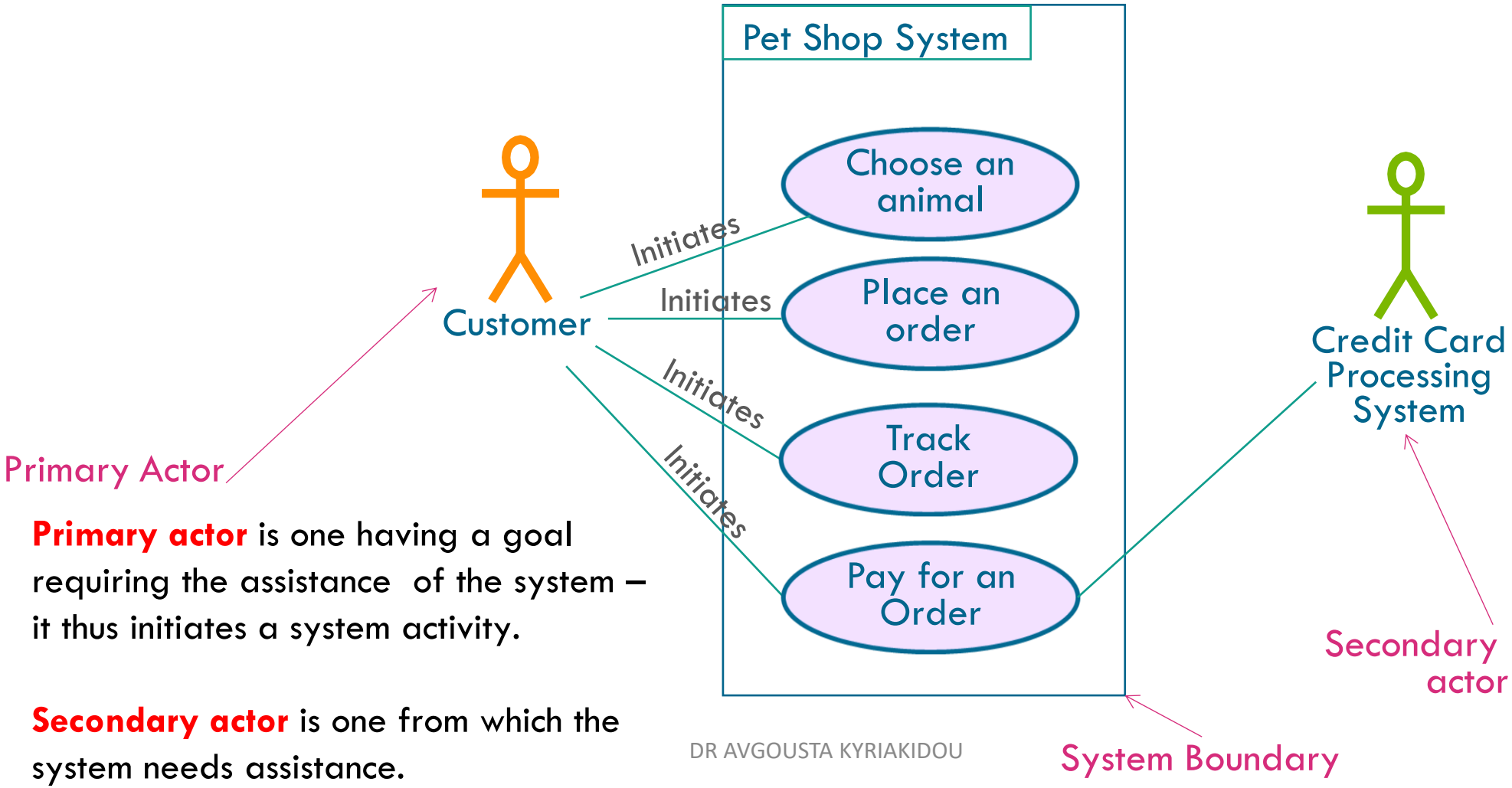






Use-Case Diagram

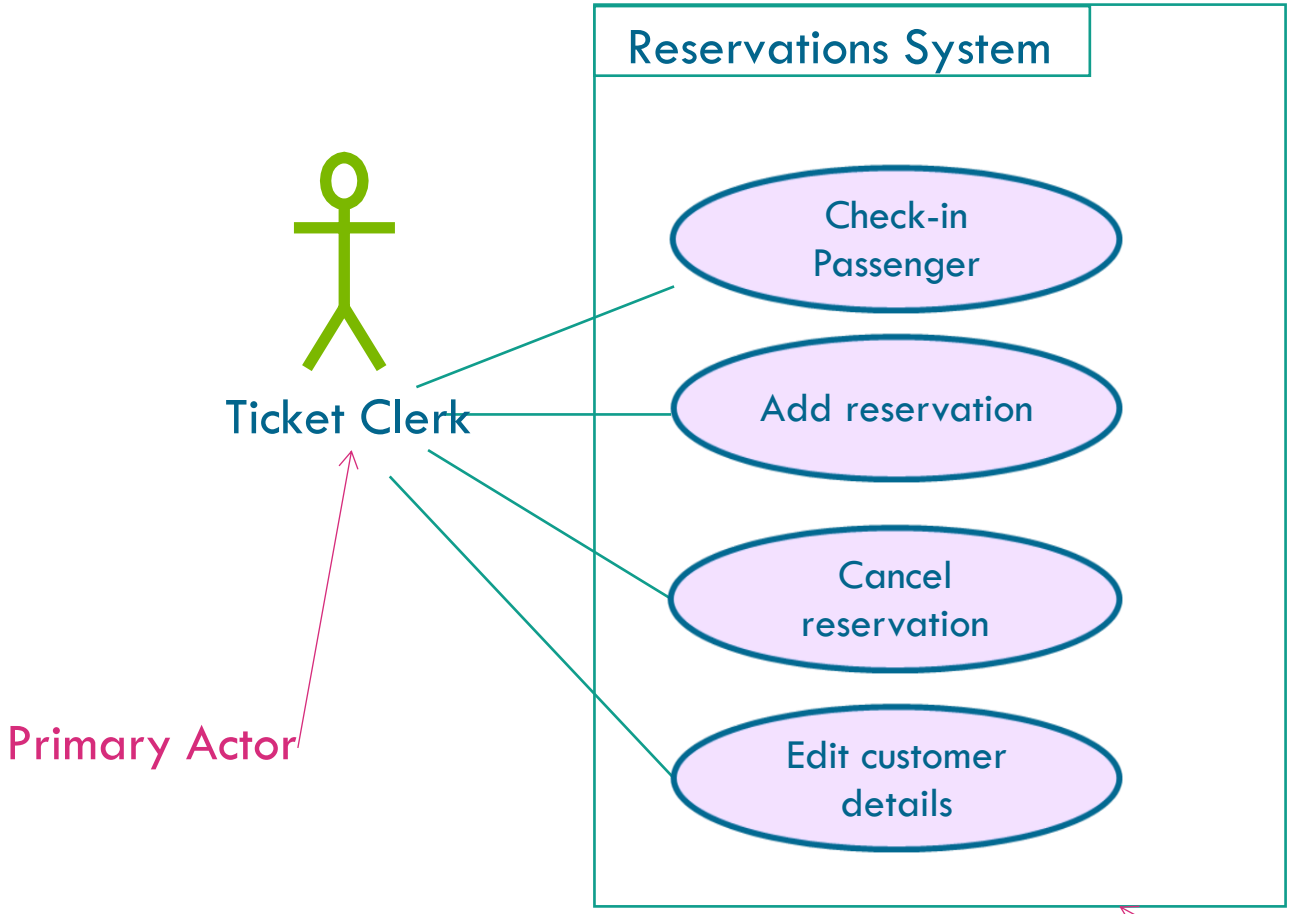
Example – Online Pet Shop

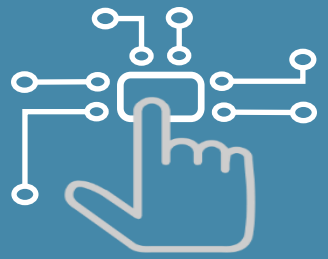


Primary actor is one having a goal requiring the assistance of the system – it thus initiates a system activity.

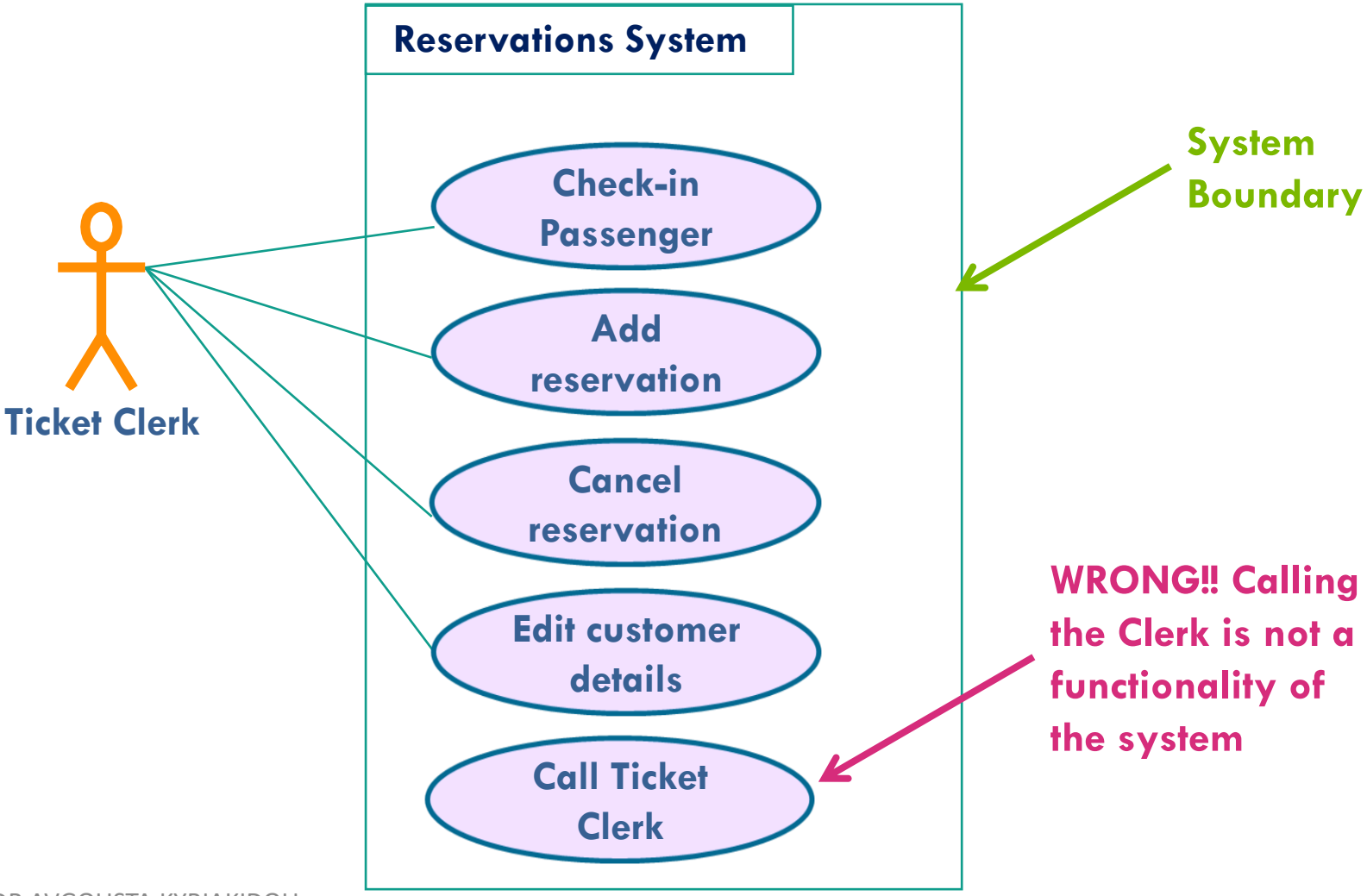
Secondary actor is one from which the system needs assistance.

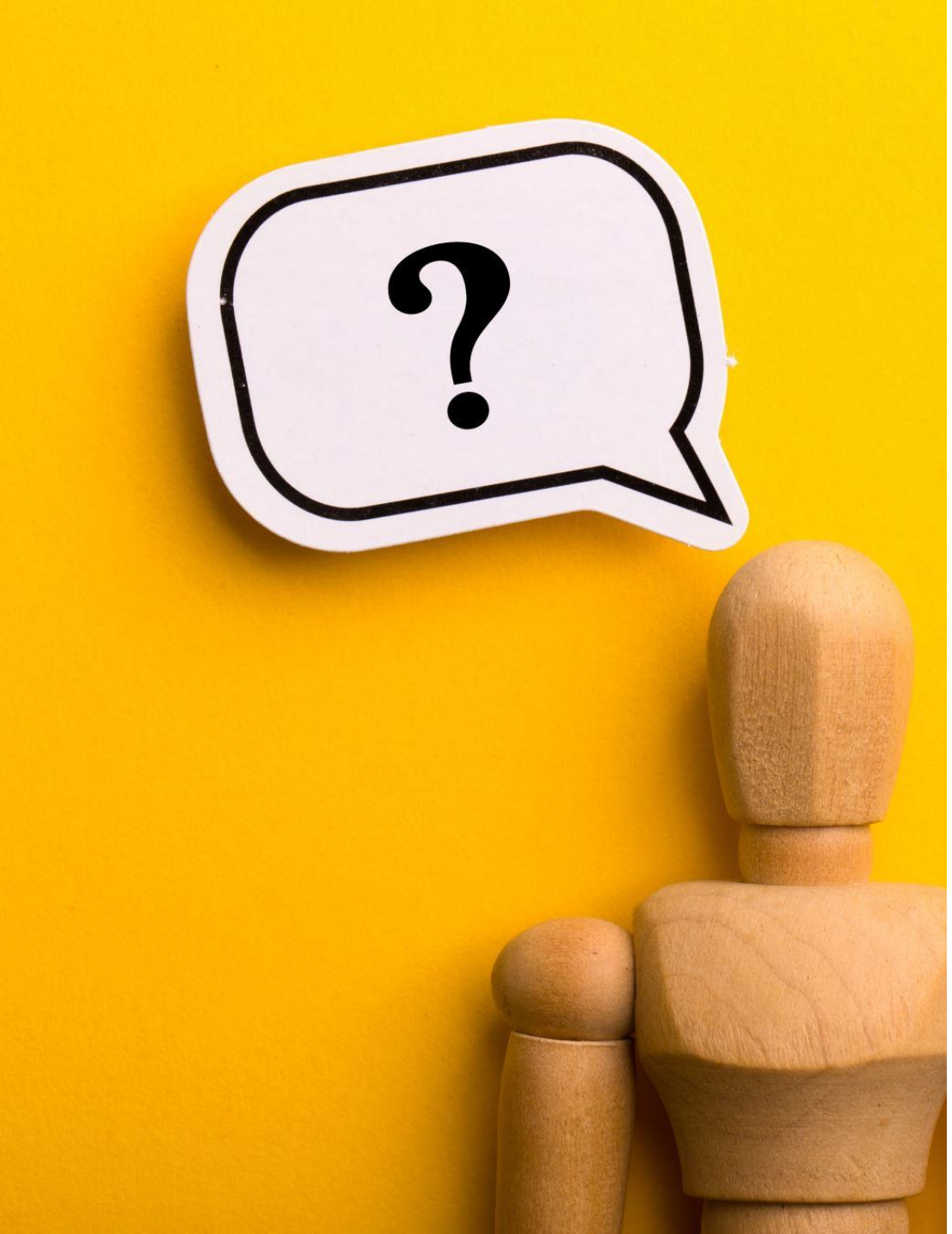
Use-Case Diagrams
Example – Reservations System 2





System Boundary
Example

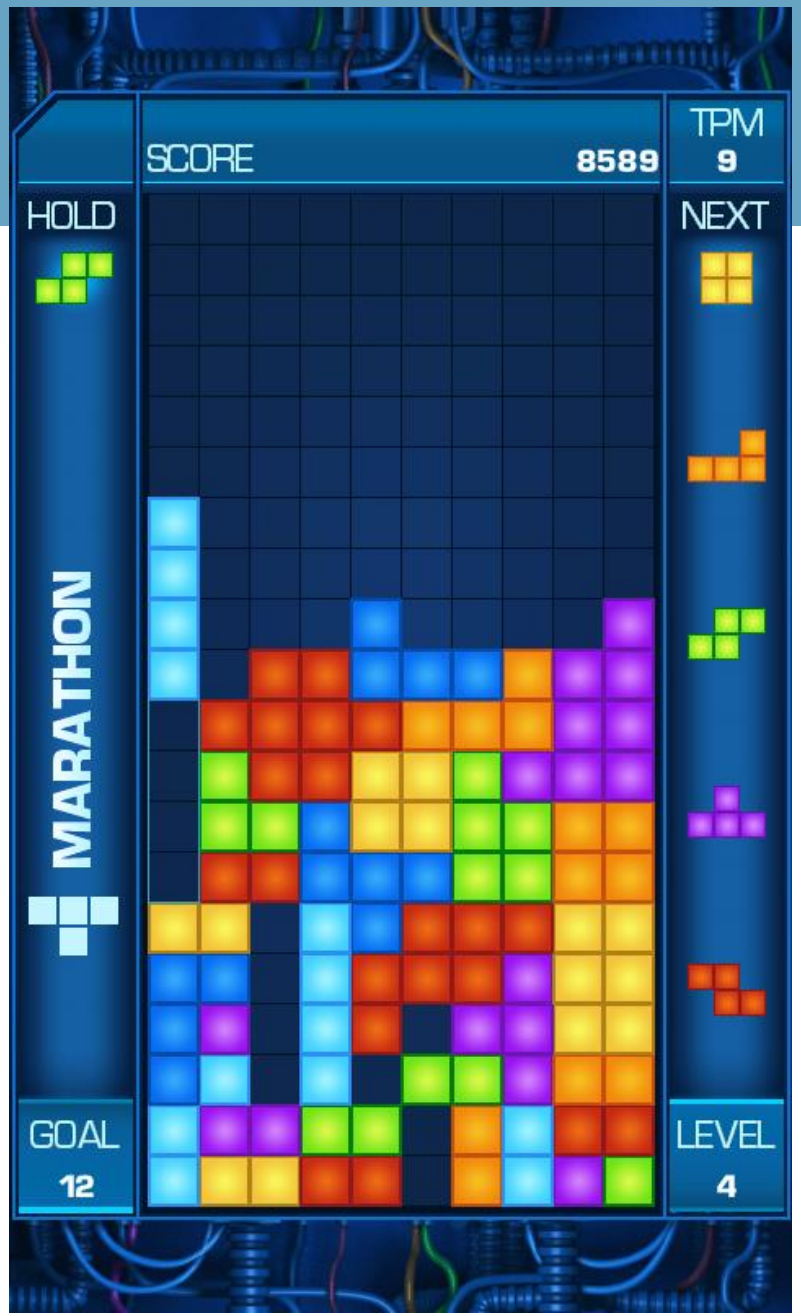




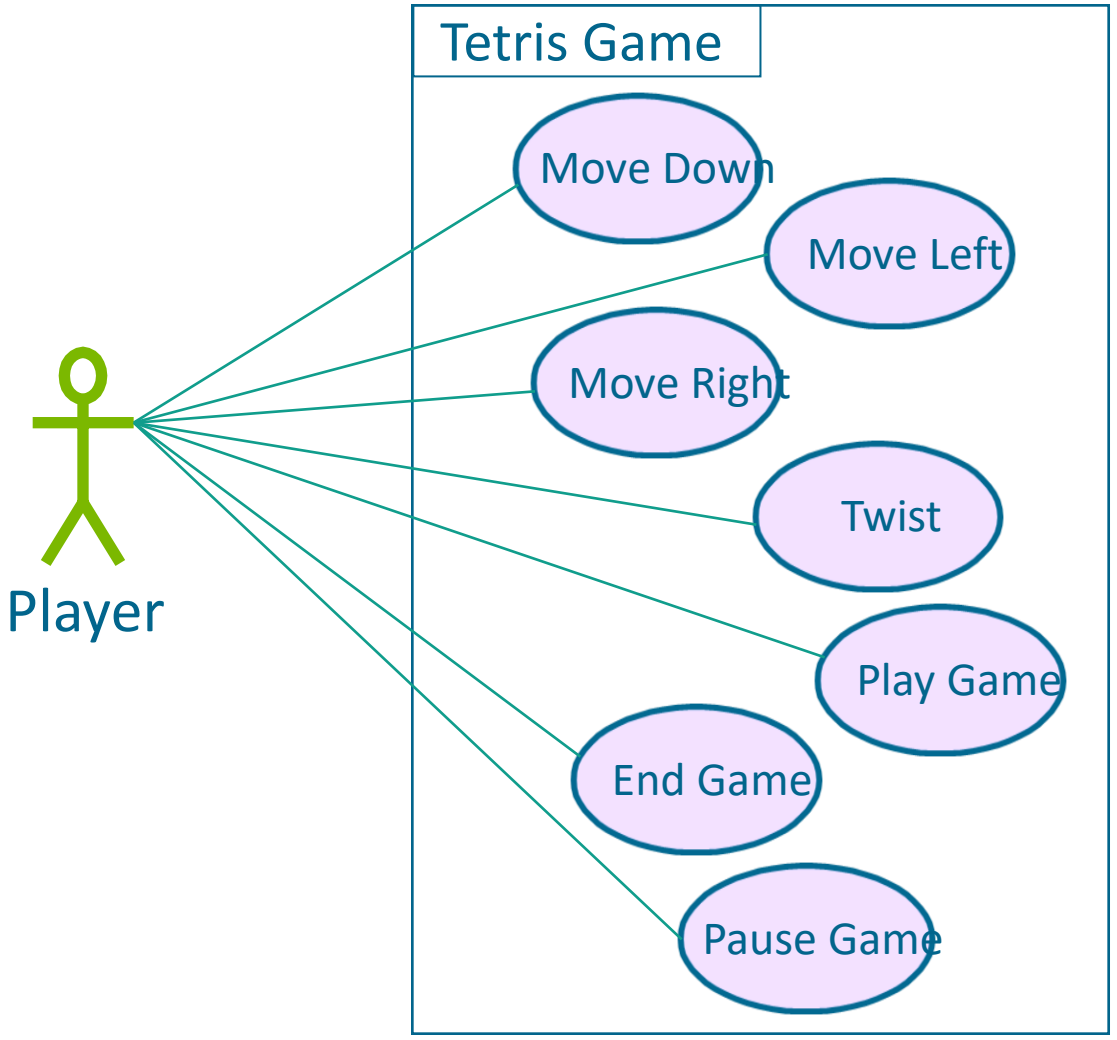
BREAKOUT SESSION

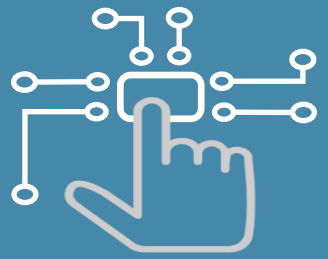
Tetris Game

- Σχεδίαστε ένα διάγραμμα περιπτώσεων χρήσης (Use-Case Diagram) για το παιχνίδι Tetris.



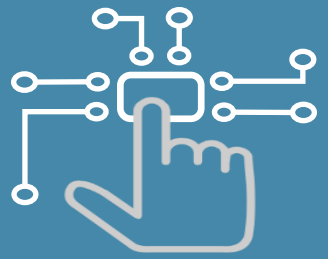
Tetris – sample basic diagram





Use-Case Diagrams
adding more design detail

- Τα διαγράμματα περιπτώσεων χρήσης αναπαριστούν τη λειτουργικότητα από πάνω προς τα κάτω - in a top-down manner .
- Το αρχικό βήμα είναι να προσδιοριστεί όλη η συμπεριφορά του συστήματος που πρόκειται να δημιουργηθεί σε ανώτατο επίπεδο (top-level behaviour)
  - Αναγνωρίζουμε τις βασικές λειτουργίες και όλα τα πράγματα που πρέπει να μπορεί να κάνει το σύστημα.
- Συνεχίζουμε να προσθέτουμε λεπτομέρεια στο διάγραμμα μας με την αποσύνθεση των προσδιορισμένων περιπτώσεων χρήσης (use-cases) σε άλλες περιπτώσεις χρήσης (use-cases) οι οποίες **<<περιλαμβάνονται>> (χρησιμοποιούνται) (included)** ή **<<επεκτείνονται>> (extended)** από τις περιπτώσεις χρήσης ανωτάτου επιπέδου
- Με αυτό τον τρόπο μπορούμε να δείξουμε τη σχέση **γενίκευσης/ειδίκευσης (generalisation/specialisation)** μεταξύ των use cases



Adding More Detail to Use-Case Diagrams
Είδη σχέσεων

- **Μεταξύ του actor και των use-case**
  - Ο Actor χρησιμοποιεί/ενεργοποιεί ένα use-case , association relationship

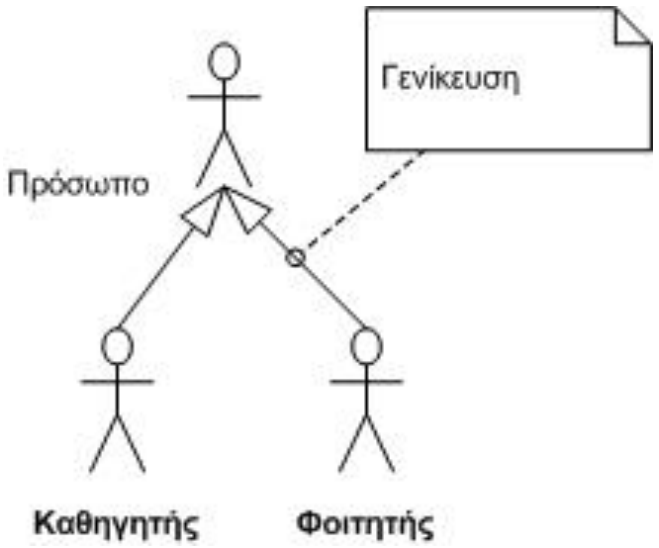
- **Μεταξύ use-cases: Στερεότυπα**
  - Τρία στερεότυπα UML
    <<extend>> το οποίο προσδιορίζει σχέση επέκτασης η οποία είναι προαιρετική
    <<include>> το οποίο προσδιορίζει σχέση συμπερίληψης η οποία είναι υποχρεωτική
  - **Σχέση γενίκευσης** μεταξύ use cases, η οποία δείχνει κληρονομικότητα (inheritance), την δείχνουμε με ένα άδειο (hollow) triangle

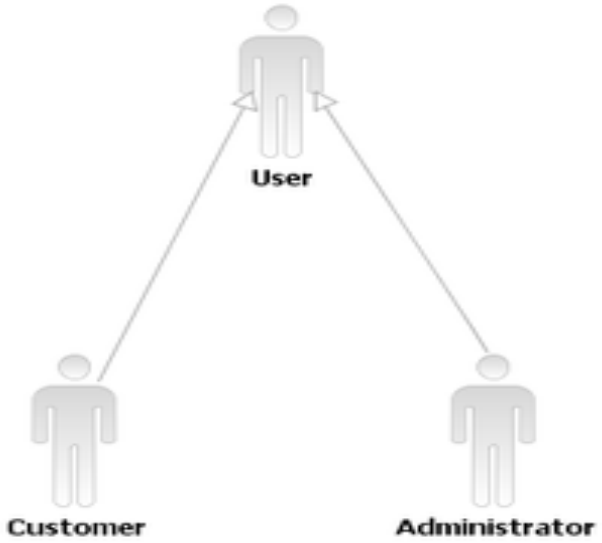
- **Μεταξύ actors: Γενίκευση (Generalisation) των actors (κληρονομικότητα- inheritance)**
  - Για να δείξουμε ότι μπορούμε να έχουμε πολλά είδη χρηστών

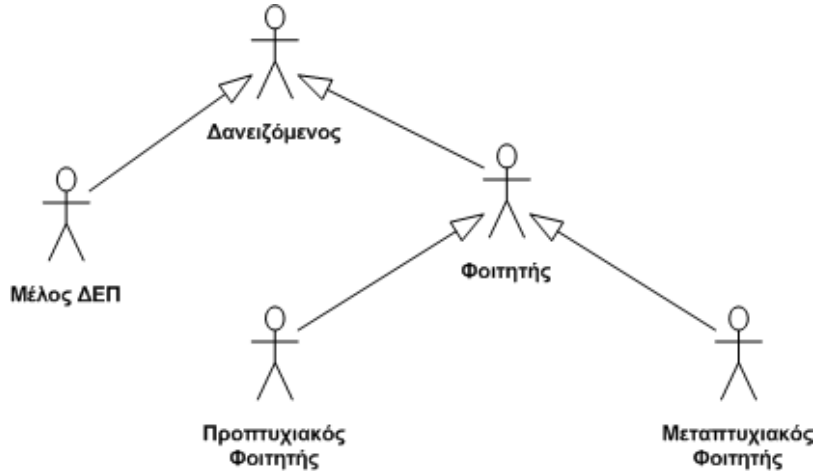
ΣΧΕΣΗ ΜΕΤΑΞΥ ACTORS - ΓΕΝΙΚΕΥΣΗ

- **Χρησιμοποιούμε τη γενίκευση των actor όταν θέλουμε να δείξουμε ομοιότητες μεταξύ των actors**
- **Σχέση Generalization/Specialization.**
- **Οι actors θα πρέπει να εμφανίζουν κοινή συμπεριφορά σε σχέση με το σύστημα**
- Ο συμβολισμός είναι μια συμπαγής γραμμή που καταλήγει σε ένα κενό τρίγωνο. Από τον εξιδεικευμένο στον γενικό actor.

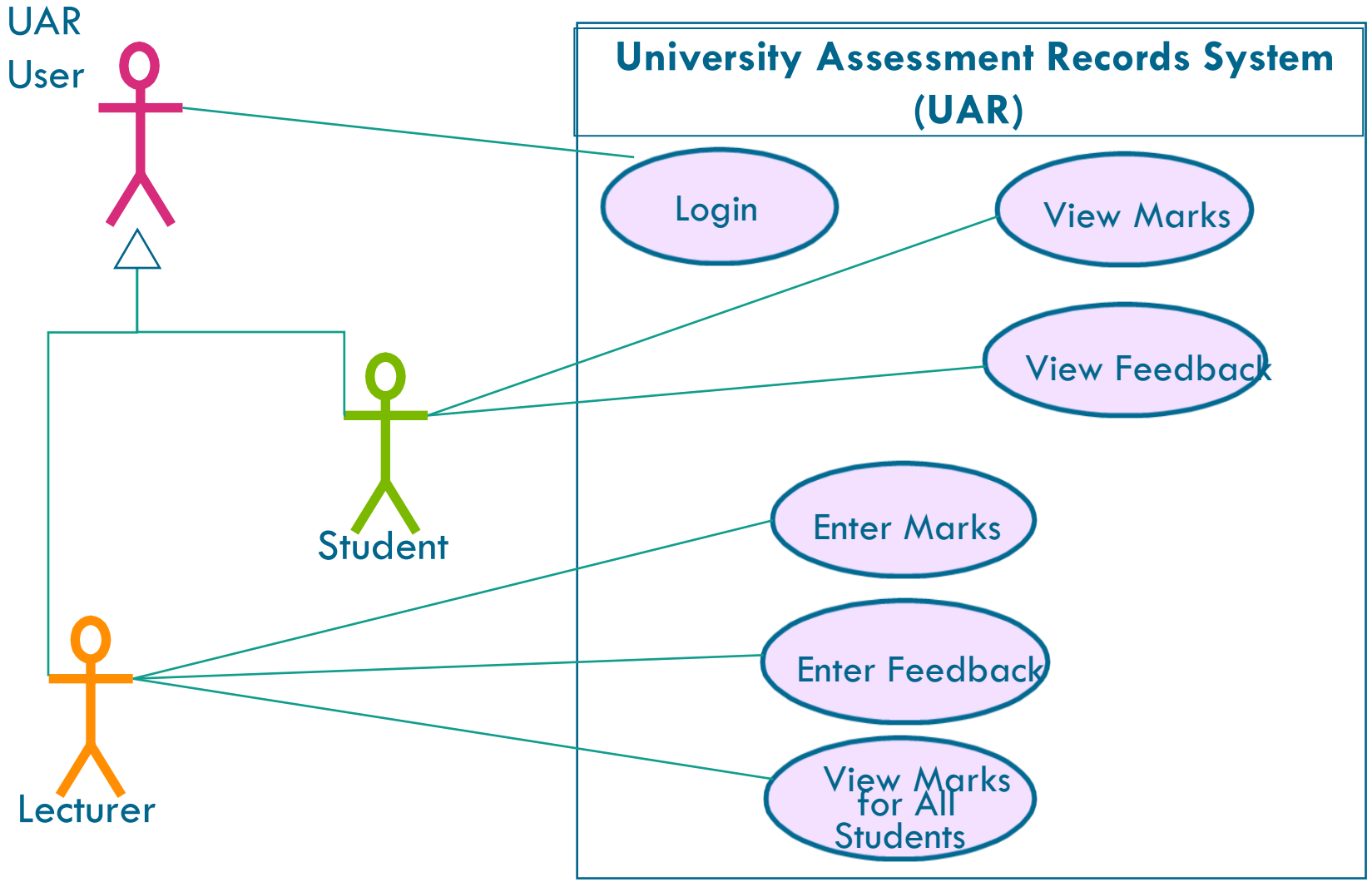
Inheritance

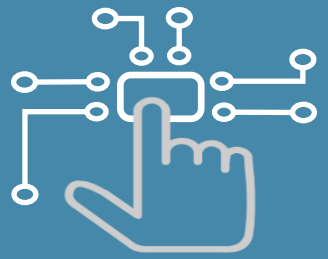






GENERALISATION OF ACTORS - EXAMPLE



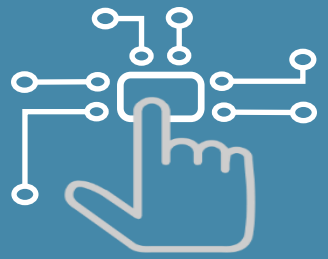


Use-Case Relationship Types
Σχέση συμπερίληψης <<includes>>

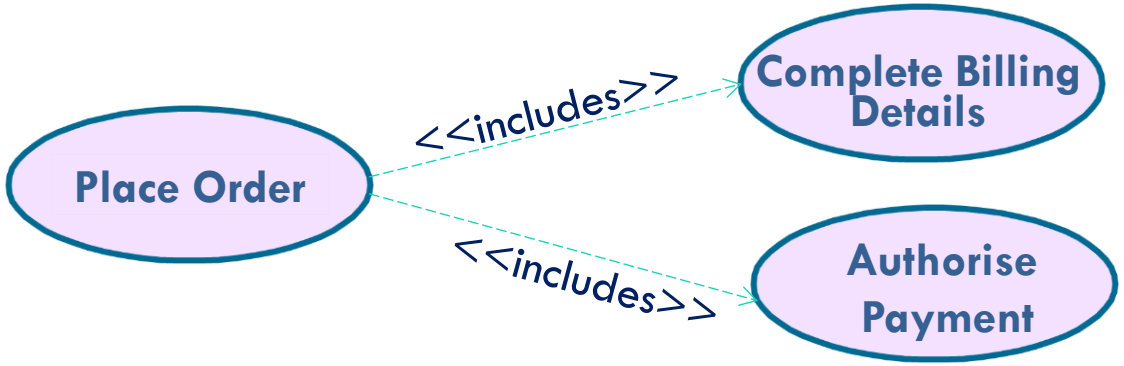
- **Στα βήματα της περίπτωσης χρήσης use-case (A) συμπεριλαμβάνονται τα βήματα της περίπτωσης χρήσης Use-Case (B)**

  - Η A αναφέρεται ως βασική και η B ως συμπεριλαμβανόμενη (included) περίπτωση χρήσης
  - Το τόξο αρχίζει από το use case A και πάει στο use case B για να δείξει ότι **"για να εκτελεστεί η λειτουργία (use case A) πρέπει η λειτουργία (use case B) να εκτελεστεί πρώτα τουλάχιστον μια φορά**."
  - **Είναι υποχρεωτική συμπερίληψη (mandatory inclusion)** – π.χ για να εκτελεστεί επιτυχώς η περίπτωση χρήσης A, η περίπτωση χρήσης B **ΠΡΕΠΕΙ ΝΑ ΟΛΟΚΛΗΡΩΘΕΙ ΕΠΙΤΥΧΩΣ τουλάχιστον μία φορά.**

- Εάν μια περίπτωση χρήσης **<<περιλαμβάνει>>(<<includes>>)** αρκετές άλλες, όλες αυτές οι άλλες περιπτώσεις χρήσης **ΠΡΕΠΕΙ ΝΑ ΟΛΟΚΛΗΡΩΘΟΥΝ ΠΡΩΤΑ**, αν και στα διαγράμματα περιπτώσεων χρήσης δεν μας ενδιαφέρει η σειρά με την οποία ολοκληρώνονται.



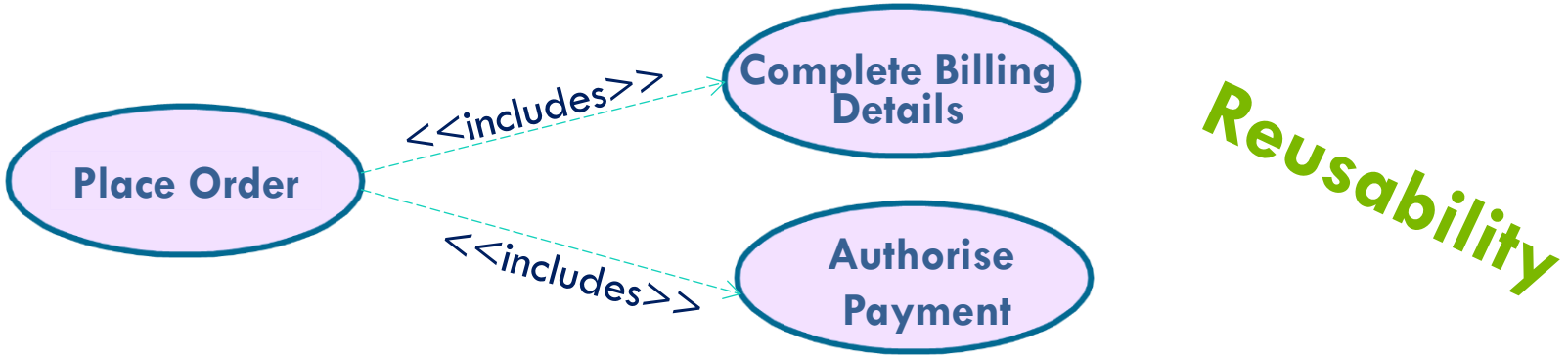
Use-Case Relationship Types <<includes>>



Reusability

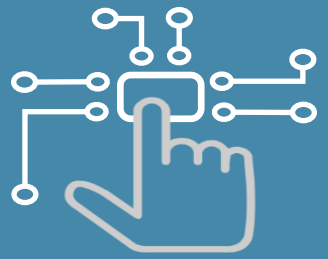
- **<<includes>> is used when:**
  - **Abstracting common behaviour (Αφαιρούμε κοινή συμπεριφορά) -** συμπεριφορά που είναι παρόμοια σε πολλές περιπτώσεις χρήσης μπορεί να διαχωριστεί σε ξεχωριστές περιπτώσεις χρήσης.
  - **Decomposing complicated behaviour that is mandatory (Αποσύνθεση περίπλοκης συμπεριφοράς που είναι υποχρεωτική)-** Ξεχωρίστε την ως ξεχωριστή περίπτωση χρήσης και αφήστε τους άλλους να την "συμπεριλάβουν"

Use-Case Relationship Types <<includes>>

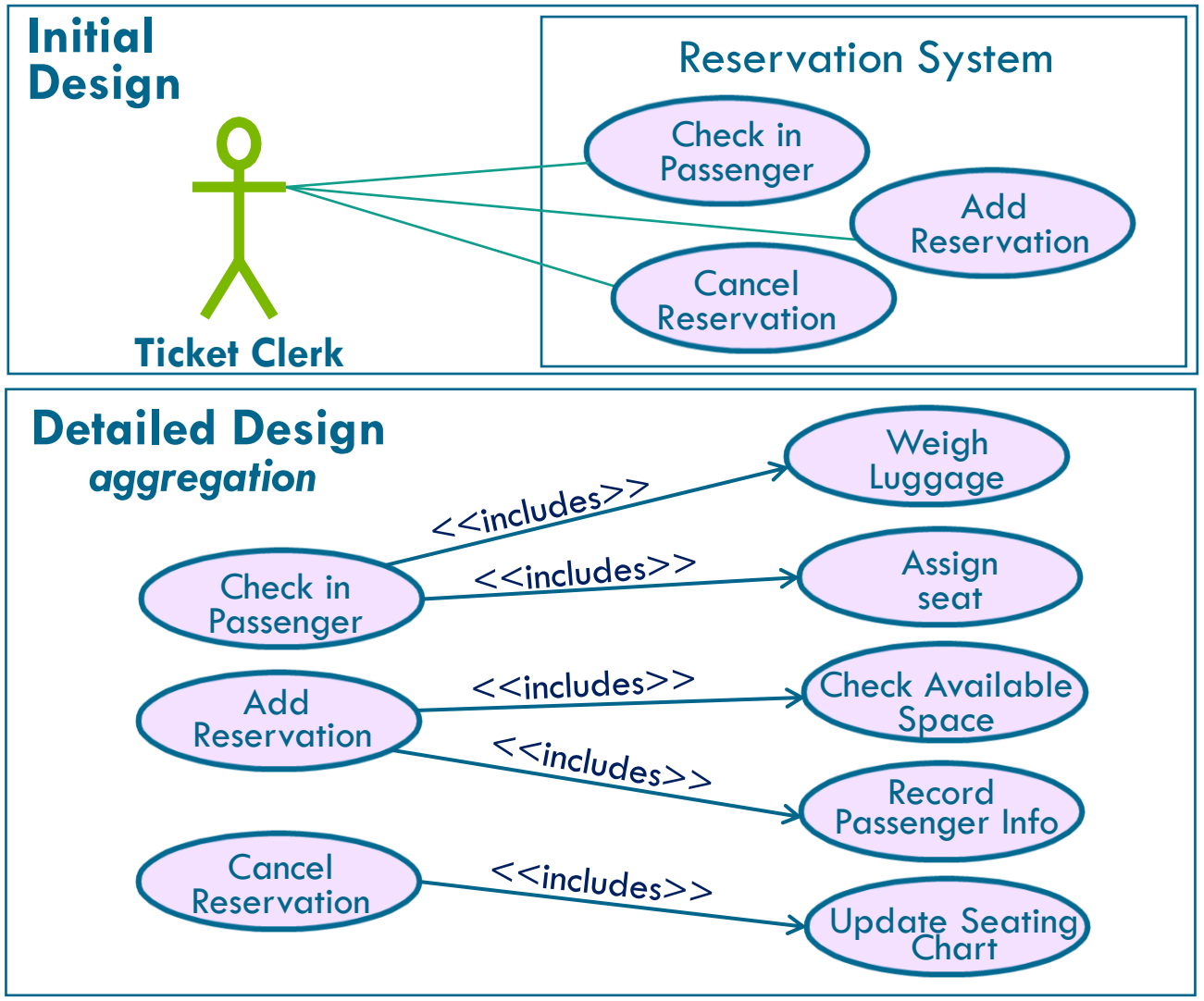


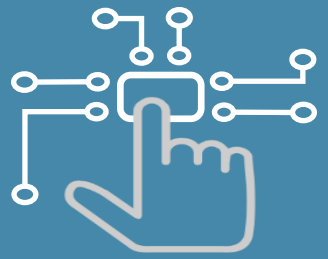
Τι σημαίνει <<includes>> πρακτικά:

- Είναι ένα use case που **εκτελείται πάντα** ως μέρος ενός άλλου use case.
- Το **κύριο use case δεν μπορεί να εκτελεστεί χωρίς** το included.
- Είναι συχνά **επαναχρησιμοποιήσιμο** (δηλ. και άλλα use cases μπορεί να το χρησιμοποιούν).
- Συνήθως **το σύστημα το εκτελεί αυτόματα**, χωρίς να απαιτείται επιπλέον ενέργεια από τον χρήστη.
- Μπορείς να σκέφτεσαι τα <<includes>> use cases ως **αυτοματοποιημένες λειτουργίες** του συστήματος, που **συμβαίνουν πάντα, εσωτερικά και υποχρεωτικά** στο πλαίσιο ενός κύριου use case



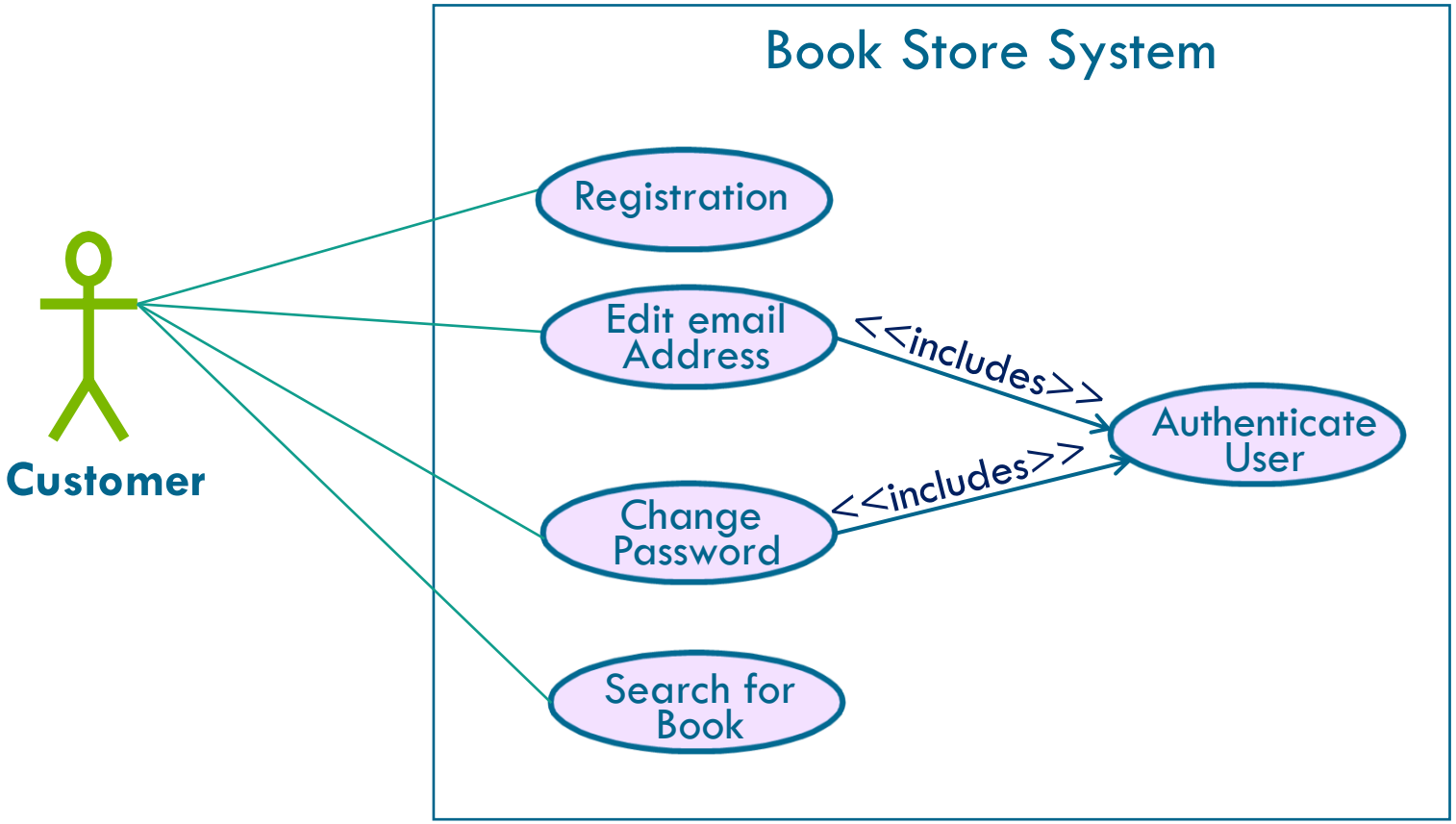
<<Includes>> example
reservations system





<<Includes>> example

book store system

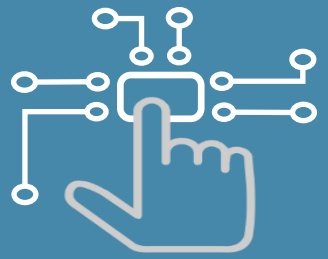


Use-Case Relationship Types
Σχέση επέκτασης <<extends>>

- Αυτή η σχέση υποδηλώνει ότι η περίπτωση χρήσης B επεκτείνει τη λειτουργικότητα της περίπτωσης χρήσης A χωρίς η A να το γνωρίζει

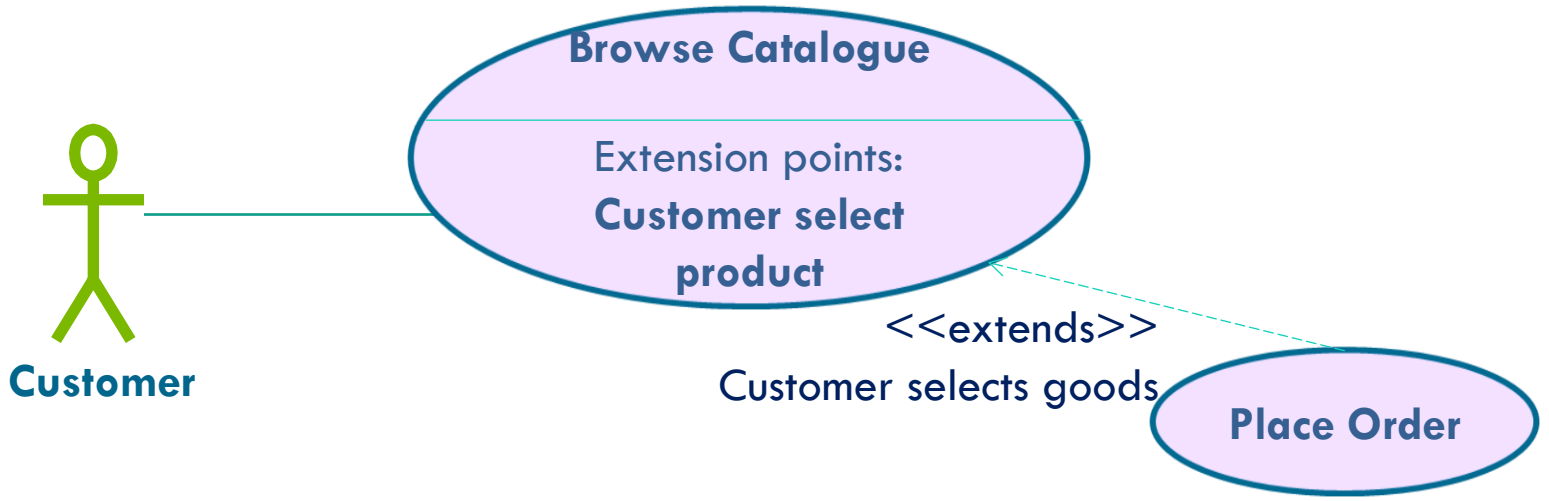
- **Το base use-case A μπορεί να ενεργοποίηση το use-case B όταν κάποια κριτήρια πληρωθούν. Αυτά τα κριτήρια ονομάζονται extension points**

1. **Υπονοεί προαιρετική επέκταση**– π.χ το use-case ***A*** μπορεί να εκτελεστεί επιτυχώς χωρίς το use case B. Το use case B επεκτείνει τη λειτουργικότητα του use case A (προσθέτει συμπεριφορά) μόνο όταν πληρούνται οι καθορισμένες συνθήκες.

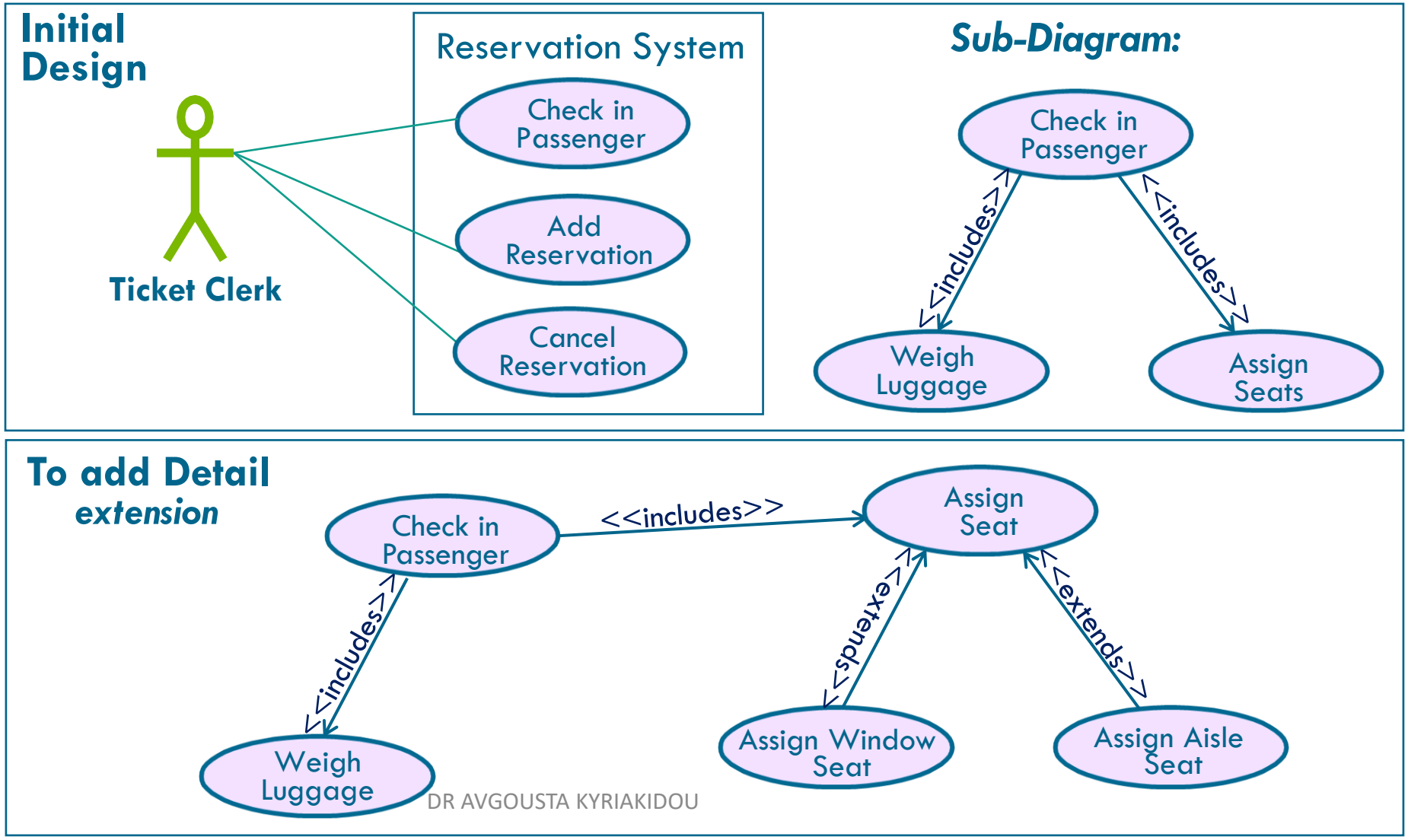


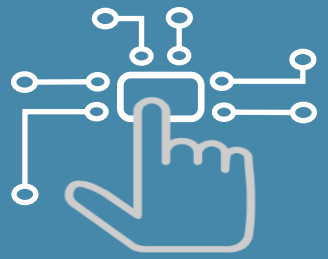
<<extends>>

Optional behaviour

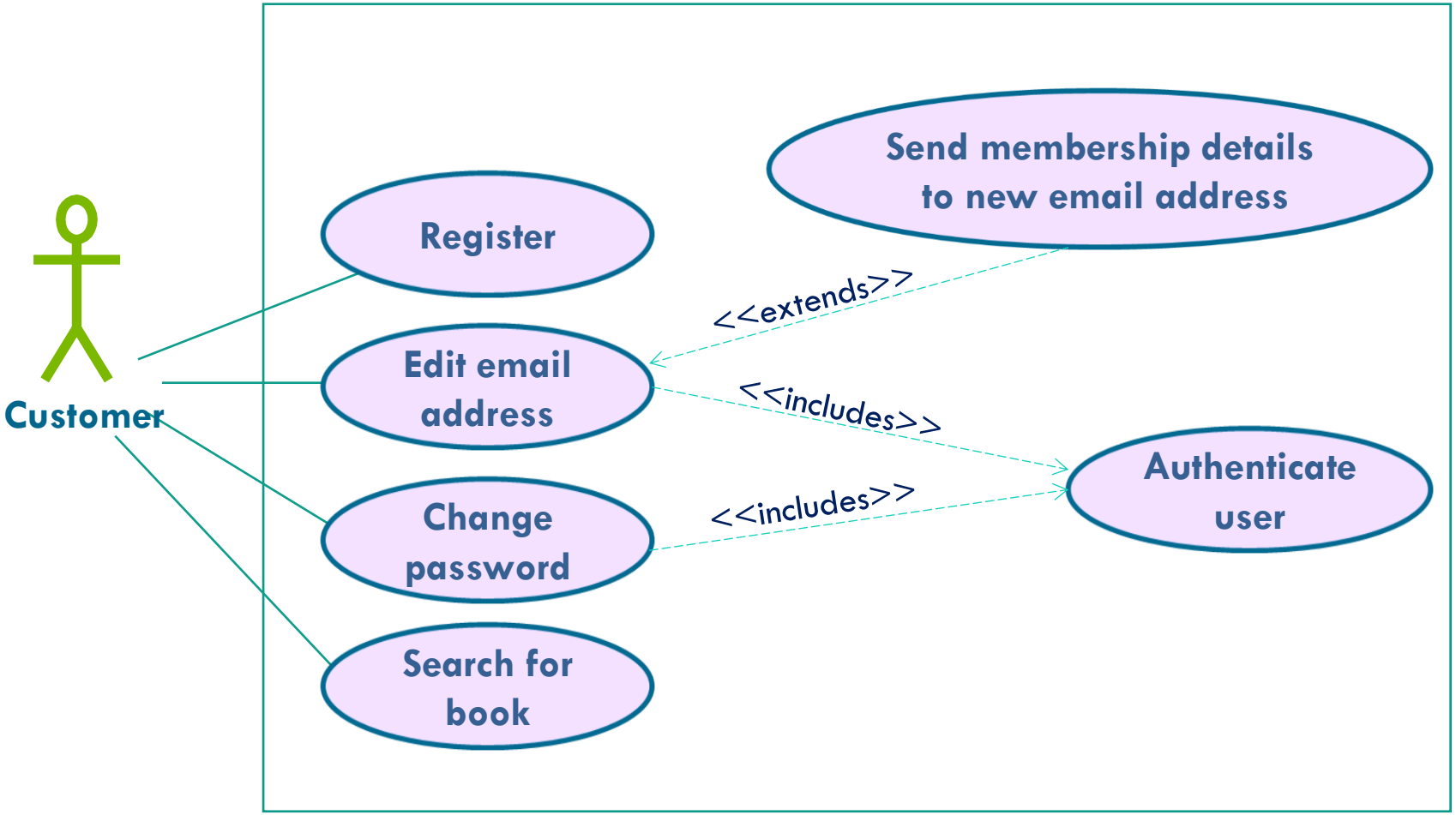


<<extends>> example
Reservations System



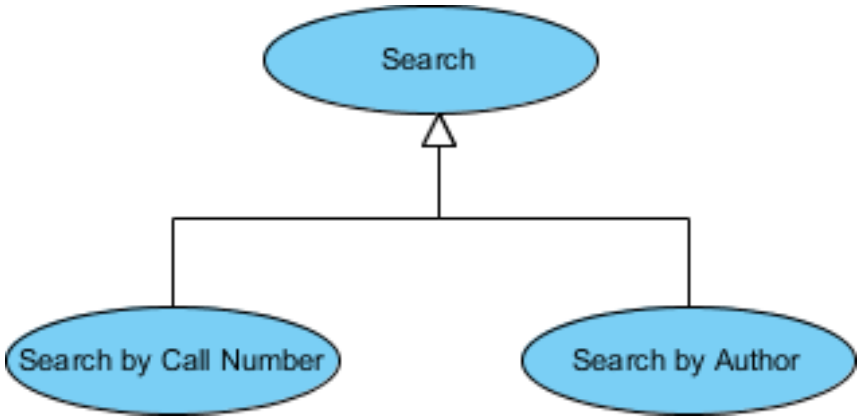


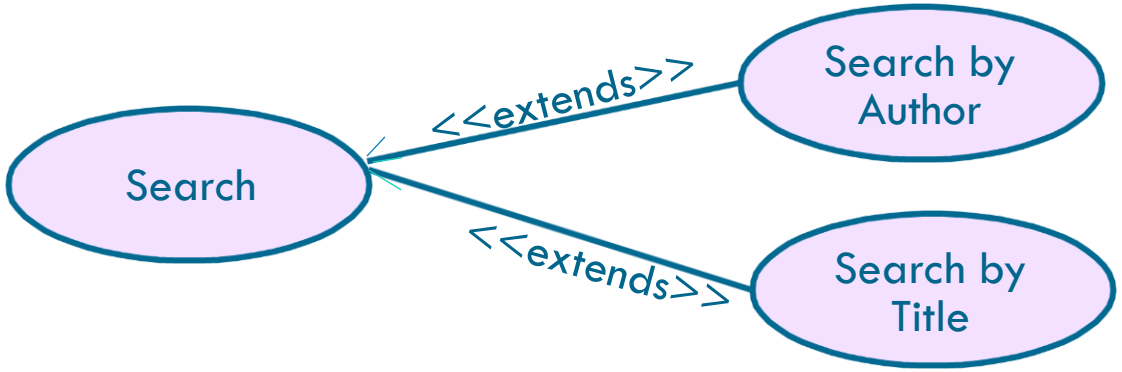
<<extends>> example
Book Store System



Note: The primary actor is only linked to the primary use case and not to the <<extends>> or <<includes>> use case

Σχέση γενίκευσης
<<generalization relationship>>

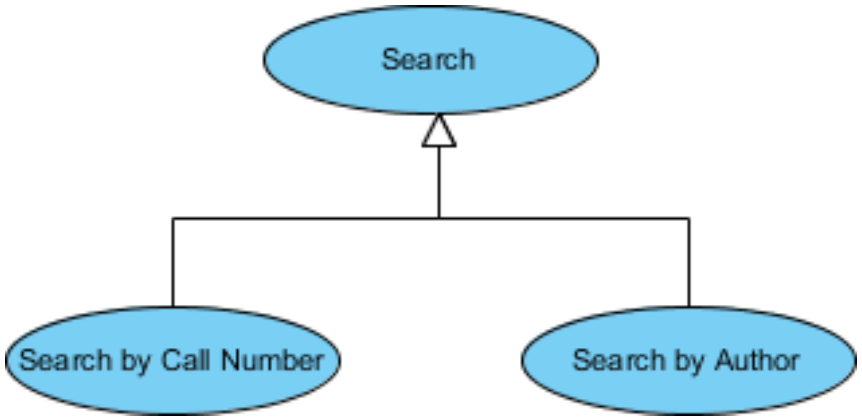


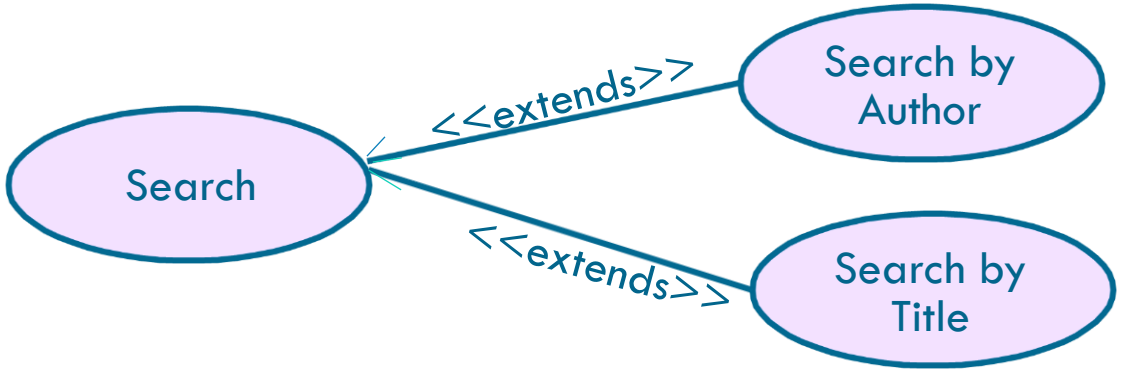


Inheritance polymorphism

- Η σχέση γενίκευσης χρησιμοποιείται όταν:
  - Μια περίπτωση χρήσης είναι παρόμοια με μια άλλη, αλλά κάνει λίγο περισσότερα.
  - Βάζουμε την κανονική/κοινή συμπεριφορά σε μία περίπτωση χρήσης
  - την ειδική συμπεριφορά σε μια ξεχωριστή περίπτωση χρήσης

Σχέση γενίκευσης
<<generalization relationship>>





Inheritance
polymorphism

Η σχέση επέκτασης <<extends>> μπορεί επίσης να χρησιμοποιηθεί για να δείξει και την σχέση γενίκευσης.

- Σχεδιάζεται από μια περίπτωση χρήσης B σε μια περίπτωση χρήσης A για να δείξει ότι η περίπτωση χρήσης B επεκτείνει τη λειτουργική συμπεριφορά της γενικότερης περίπτωσης χρήσης A.

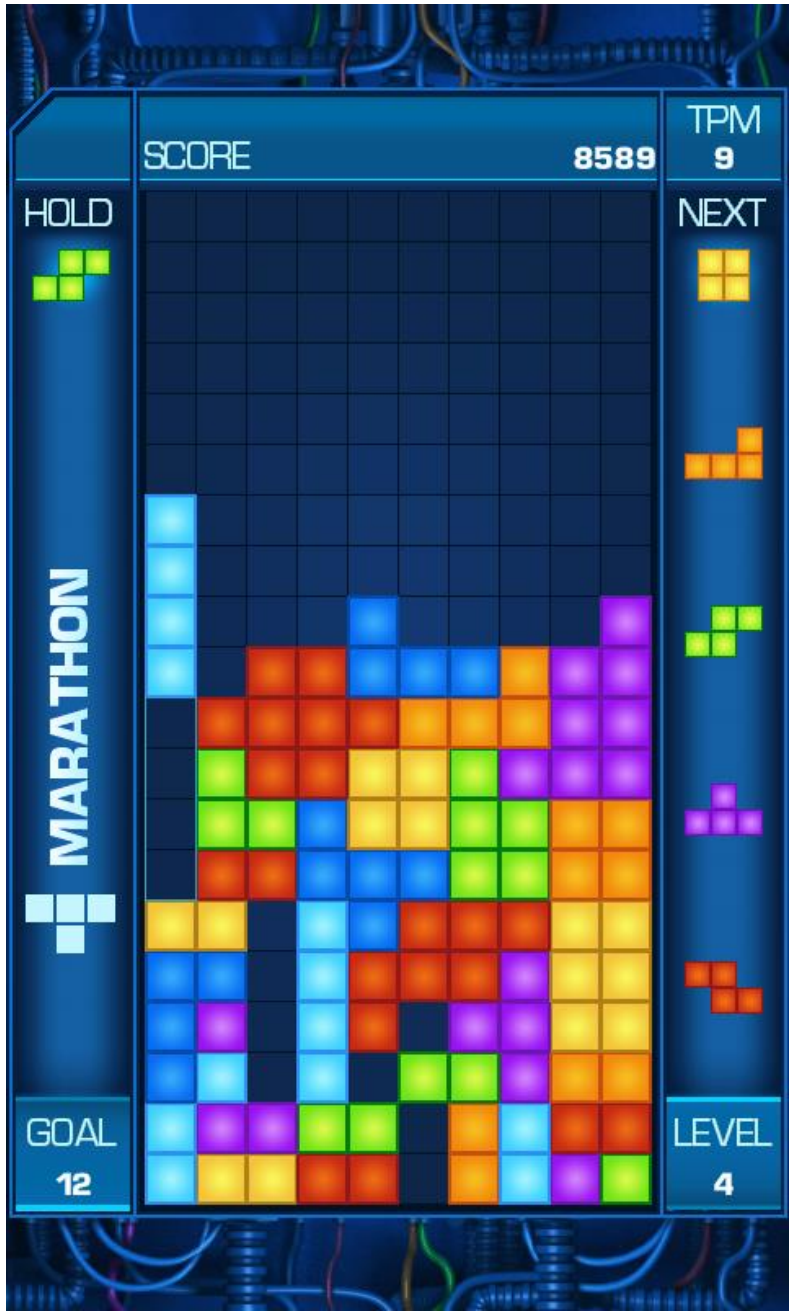
Ωστόσο, στις περισσότερες περιπτώσεις αυτή η συμπεριφορά θα μπορούσε επίσης να καταδειχθεί οπτικά με μια σημειογραφία κληρονομικότητας μεταξύ των περιπτώσεων χρήσης, το κενό τρίγωνο από τις γενικές στις εξειδικευμένες περιπτώσεις χρήσης.

BREAKOUT SESSION

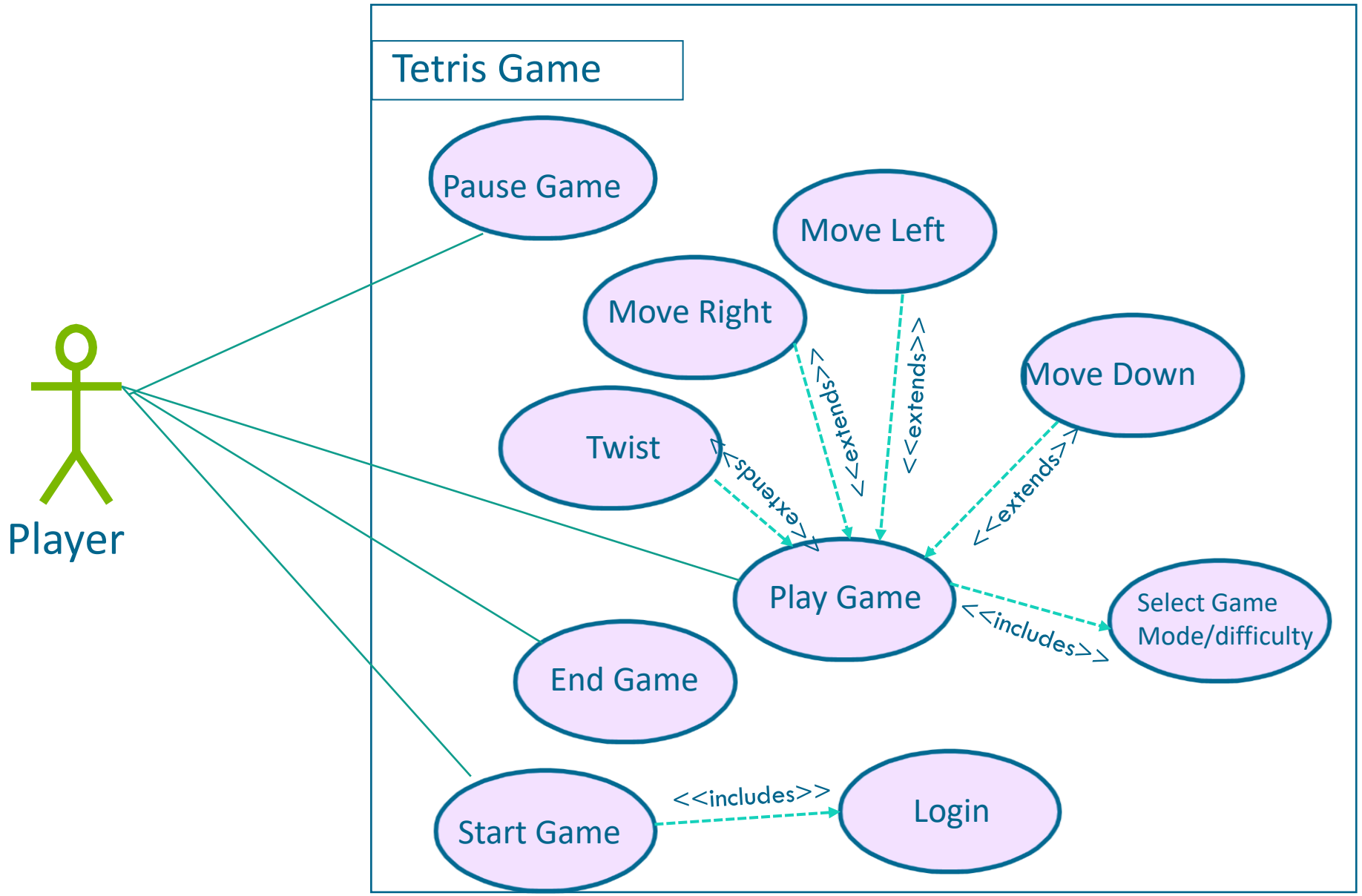
DR AVGOUSTA KYRIAKIDOU

TETRIS GAME

- In groups of 3-4
- **Draw** a use-case diagram for the Tetris game
- Add more design detail to your previous diagram with includes and extends



TETRIS GAME SAMPLE USE CASE DIAGRAM



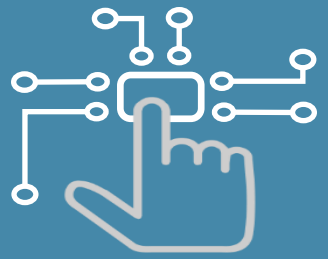
ΤΕΚΜΗΡΙΩΣΗ ΠΕΡΙΠΤΩΣΕΩΝ ΧΡΗΣΗΣ – ΣΕΝΑΡΙΑ (SCENARIOS)

Τα **σενάρια (use case scenarios)** είναι μια ακολουθία βημάτων για το πώς μπορεί να χρησιμοποιηθεί μια συγκεκριμένη λειτουργικότητα του συστήματος από τον τελικό χρήστη (ή, σε ορισμένες περιπτώσεις, από άλλο σύστημα λογισμικού).

- **Περιλαμβάνεται ως μέρος της περιγραφής περιπτώσεων χρήσης**
  - Κάθε περίπτωση χρήσης πρέπει να περιγράφεται σε ένα έγγραφο ροής γεγονότων ή σεναρίου.
  - Το scenario ορίζει τι πρέπει να κάνει το σύστημα όταν ο δράστης ξεκινά μια περίπτωση χρήσης.

ΠΕΡΙΕΧΟΜΕΝΑ ΠΕΡΙΠΤΩΣΕΩΝ ΧΡΗΣΗΣ

- Τα βήματα των περιπτώσεων χρήσης περιγράφονται με απλές καταφατικές και σύντομες προτάσεις.
- Διατυπώνουν με ακρίβεια για το τι κάνει το σύστημα και τι ο πρωτεύων actor.
- Δεν περιγράφεται το πώς δουλεύει το σύστημα αλλά μόνο το τι κάνει.
- Δεν περιγράφονται στοιχεία της διεπαφής χρήστη, όπως και άλλα στοιχεία που αφορούν τη σχεδίαση του λογισμικού.



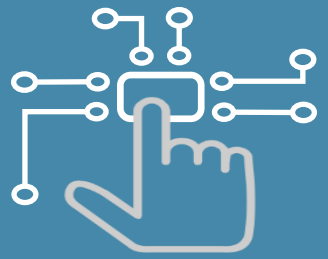
Documenting Use-Cases
Πως δημιουργούμε τα σενάρια

Για κάθε περίπτωση χρήσης, καταγράψτε την περιγραφή της. Αυτό περιλαμβάνει την παροχή των ακόλουθων πληροφοριών

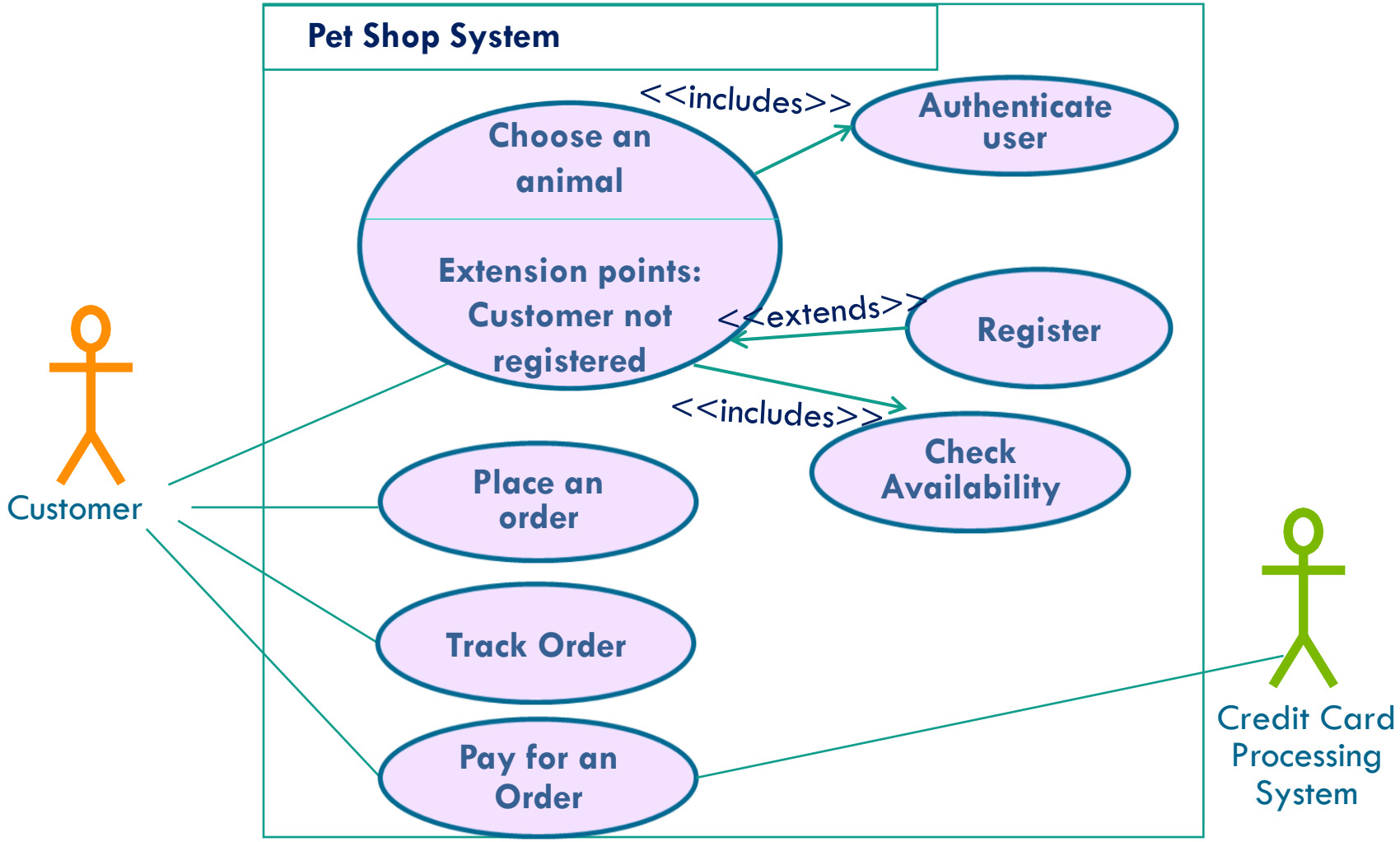
1. **Use-case name**
2. **Actors** involved (Primary actors, αυτούς που ενεργοποιούν/initiate το use case)
3. **Brief description (Σύντομη περιγραφή)**
4. **Typical course of events** περιλαμβάνει την κύρια ροή γεγονότων (δράση του actor και απόκριση του συστήματος) (actor action and system response)
5. **Alternative course of events** εναλλακτικές ροές που είναι εναλλακτικές επιτυχημένες ή αποτυχημένες ροές εκτέλεσης της περίπτωσης χρήσης.
6. **Preconditions** προϋποθέσεις που είναι απαραίτητες για την έναρξη της περίπτωσης χρήσης.
7. **Post conditions** μετα-συνθήκες που καθορίζουν την κατάσταση του συστήματος μετά το τέλος της περίπτωσης χρήσης.
8. **Assumptions** – οποιεσδήποτε άλλες εικασίες μπορεί να κάνουμε

DOCUMENTING USE CASES - WRITING THE SCENARIO

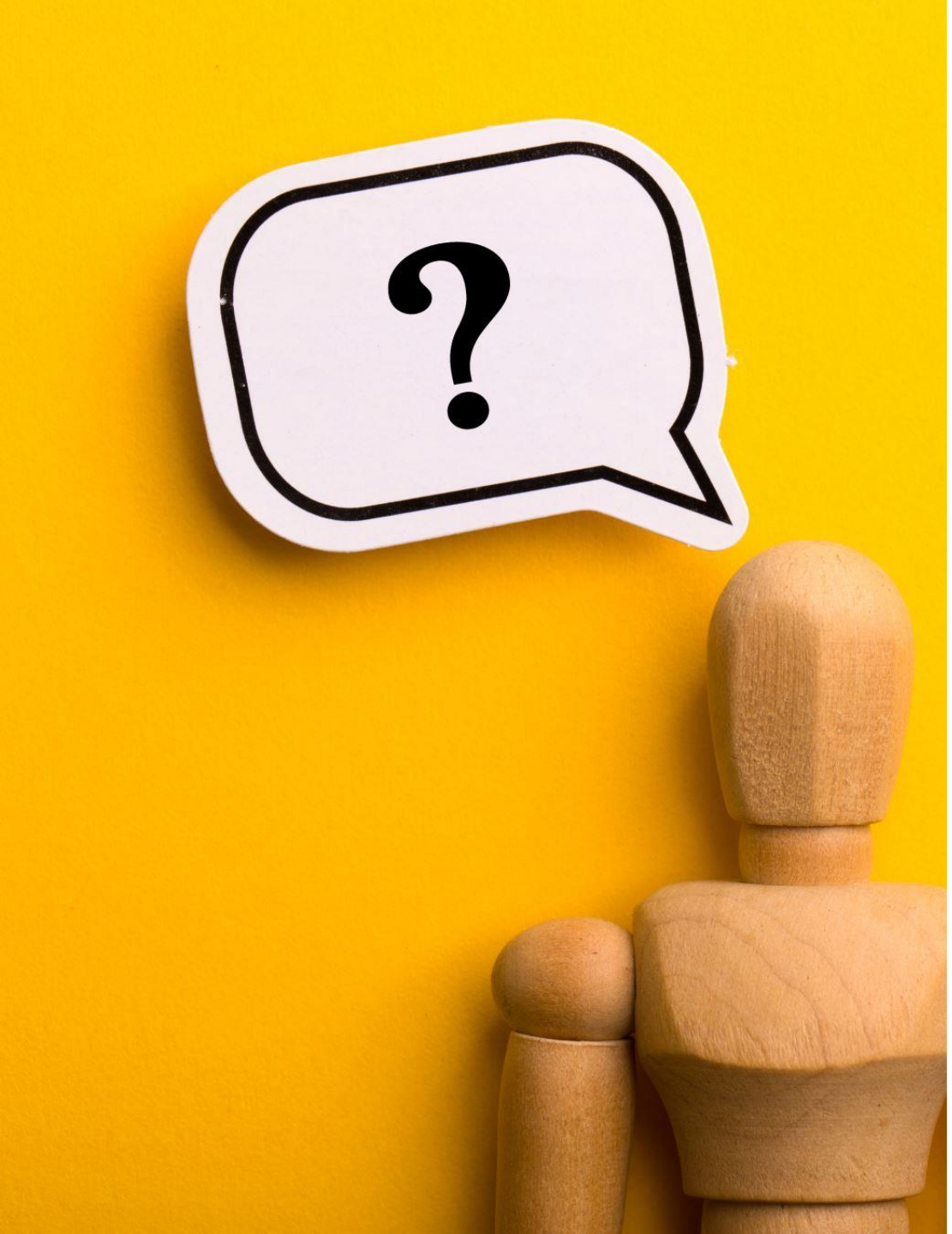
Use case Name:		
Actors:		
Description:		
Typical course of events:	Actor Action:	System Response:
Alternate course of events:		
Precondition:		
Postcondition:		
Assumptions		



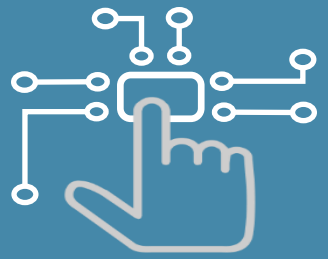
Example
Pet Shop



Use case name	Choose an animal	
Actors	Customer	
Description	This describes how a customer can select a particular animal from the website	
Typical course of events	**Actor Action** **Step 1:**This use-case is initiated by a customer **Step 3:** Customer provides username/password **Step 5:** Customer selects an animal from the list **Step 7:** Customer receives confirmation	**System response** **Step 2:** Invoke <<includes>> use-case Authorise User. **Step 4:** If correct, Customer is presented with a list of animals **Step 6:** Invoke <<includes>> use case check availability
Alternate courses	Step 2: If the customer is not already registered, then Invoke <<extends>> use-case, Register. Step 4: If username/password incorrect, then inform the customer and start the process again Step 6: If the customer chooses an animal that is not available, then a notification is sent to the customer	
Precondition	This use-case is initiated by customers	
Post condition	confirmation	
Assumptions	None at this time	

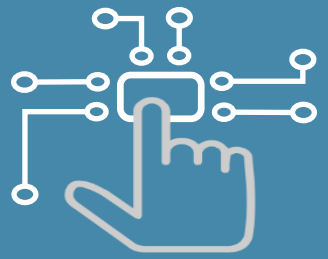


BREAKOUT SESSION



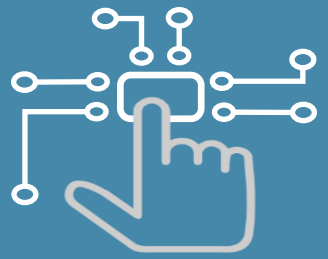
Exercise 1
University online Library system Case Study1

- Κάθε φοιτητής(student) μπορεί να δανειστεί έως και 3 βιβλία ταυτόχρονα.
- Οι φοιτητές μπορούν να περιηγηθούν σε έναν ηλεκτρονικό κατάλογο βιβλίων (browse an online catalogue for books).
- Οι δανεισμοί(Loans) πρέπει να καταγράφονται για συγκεκριμένα αντίτυπα βιβλίων και όχι απλώς για τους τίτλους τους, καθώς μπορεί να υπάρχουν πολλαπλά αντίτυπα του ίδιου βιβλίου.
- Εάν όλα τα αντίτυπα ενός βιβλίου (book copies) είναι δανεισμένα, ο φοιτητής μπορεί να κάνει κράτηση.
- Η κράτηση(reservation) καταχωρίζεται στον τίτλο του βιβλίου. Ένας φοιτητής μπορεί επίσης να επεκτείνει τον δανεισμό του, εφόσον το βιβλίο δεν έχει ήδη κρατηθεί από άλλον.
- Όταν ένα αντίτυπο του συγκεκριμένου βιβλίου γίνει διαθέσιμο, ο βιβλιοθηκάριος το δεσμεύει(reserve) για τον φοιτητή και ο φοιτητής ειδοποιείται ότι το αντίτυπό του είναι έτοιμο για παραλαβή.



Exercise 1
University online Library system Case Study1

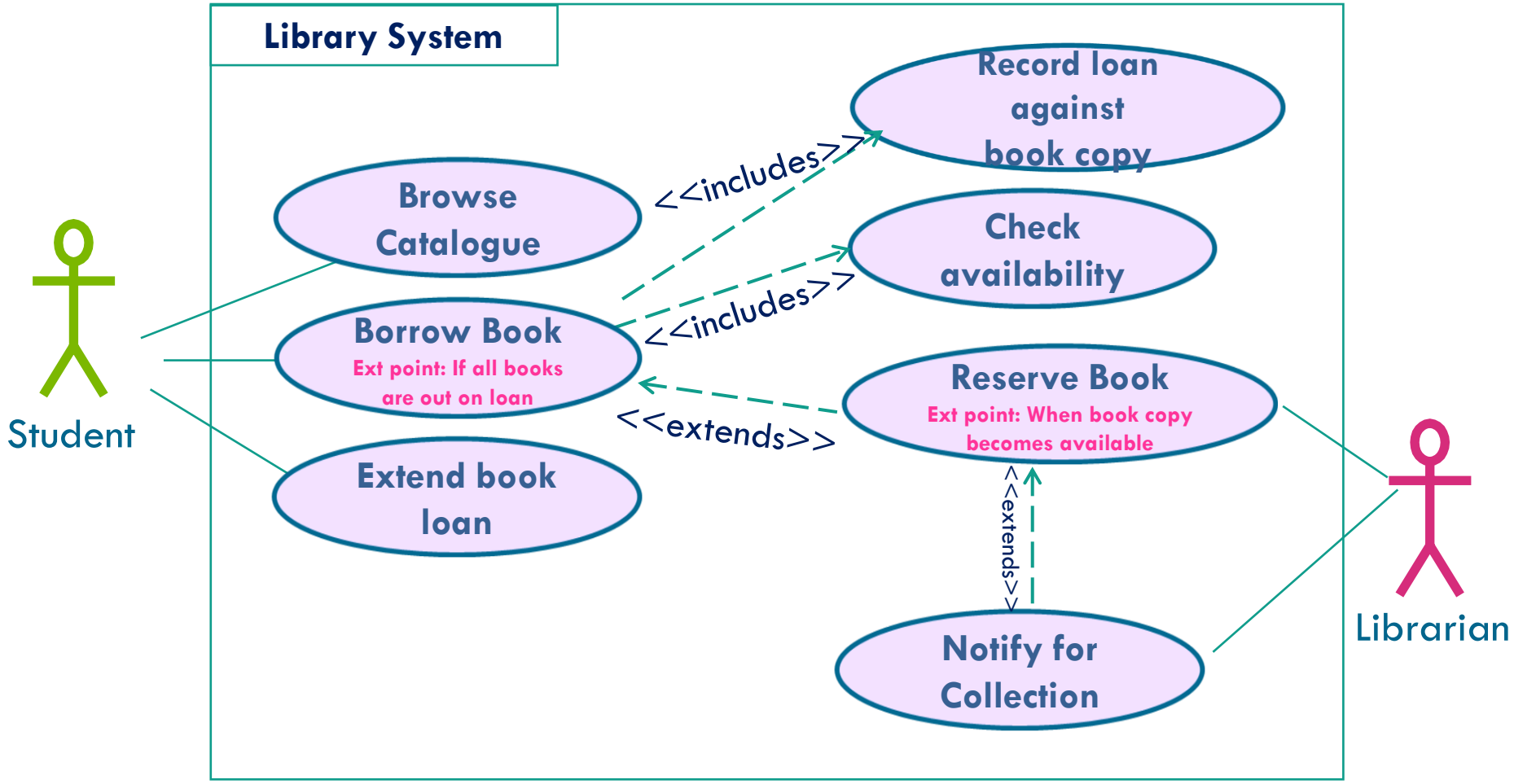
- Κάθε **φοιτητής(student)** μπορεί να δανειστεί έως και 3 βιβλία ταυτόχρονα.
- Οι φοιτητές μπορούν να περιηγηθούν σε έναν ηλεκτρονικό κατάλογο βιβλίων (browse an online catalogue for books).
- Οι δανεισμοί(Loans) πρέπει να καταγράφονται για συγκεκριμένα αντίτυπα βιβλίων και όχι απλώς για τους τίτλους τους, καθώς μπορεί να υπάρχουν πολλαπλά αντίτυπα του ίδιου βιβλίου.
- Εάν όλα τα αντίτυπα ενός βιβλίου (book copies) είναι δανεισμένα, ο φοιτητής μπορεί να κάνει κράτηση.
- Η κράτηση(reservation) καταχωρίζεται στον τίτλο του βιβλίου. Ένας φοιτητής μπορεί επίσης να επεκτείνει τον δανεισμό του, εφόσον το βιβλίο δεν έχει ήδη κρατηθεί από άλλον.
- Όταν ένα αντίτυπο του συγκεκριμένου βιβλίου γίνει διαθέσιμο, ο **βιβλιοθηκάριος** το δεσμεύει(reserve) για τον φοιτητή και ο φοιτητής ειδοποιείται ότι το αντίτυπό του είναι έτοιμο για παραλαβή.

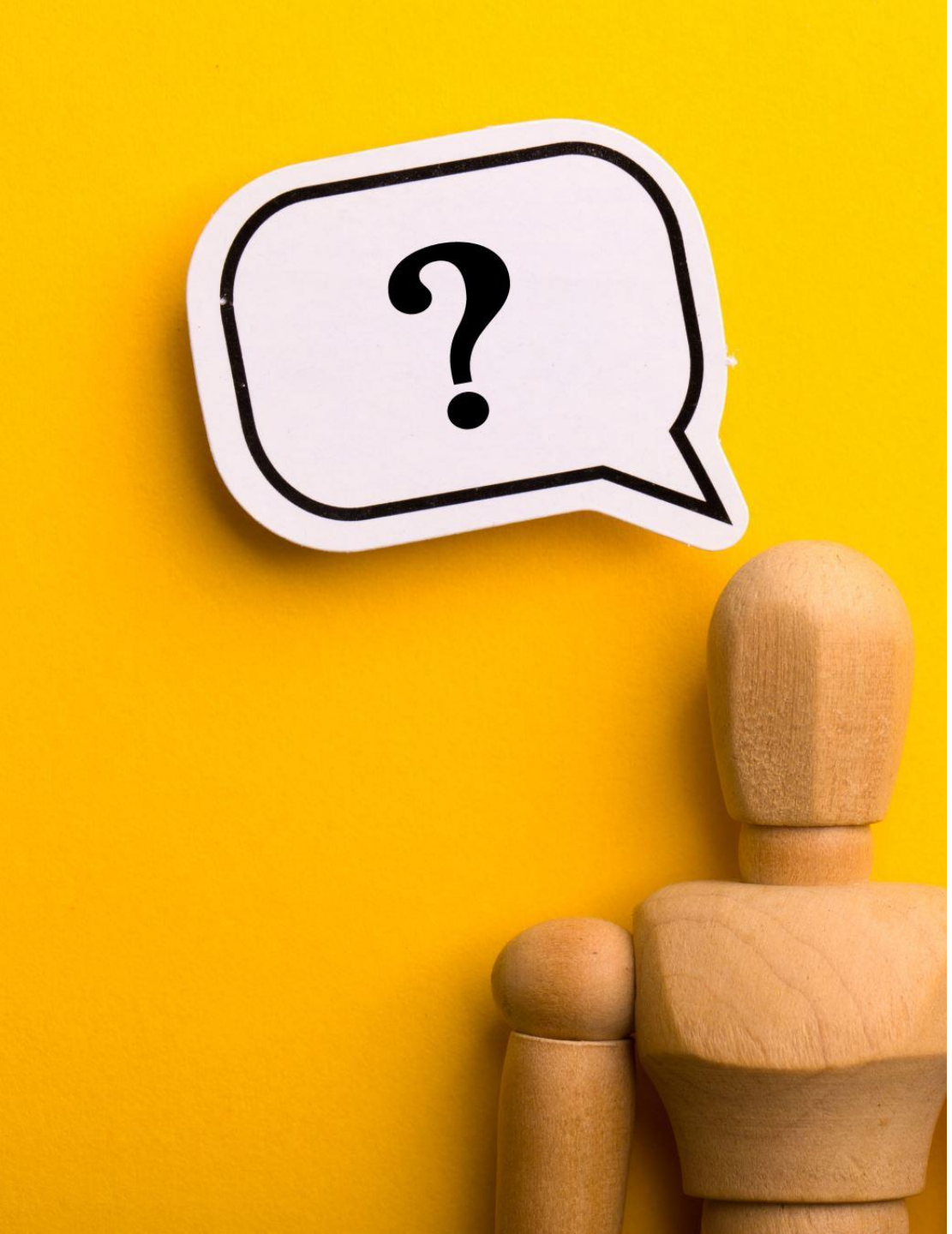


Exercise 1
University online Library system Case Study1

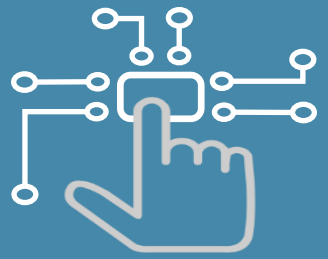
- Κάθε **φοιτητής(student)** μπορεί να **δανειστεί (borrow)** έως και 3 βιβλία ταυτόχρονα.
- Οι φοιτητές μπορούν να περιηγηθούν σε έναν ηλεκτρονικό κατάλογο βιβλίων (**browse** an **online catalogue** for books).
- Οι δανεισμοί(Loans) πρέπει να καταγράφονται για συγκεκριμένα αντίτυπα βιβλίων και όχι απλώς για τους τίτλους τους, καθώς μπορεί να υπάρχουν πολλαπλά αντίτυπα του ίδιου βιβλίου.
- Εάν όλα τα αντίτυπα ενός βιβλίου (book copies) είναι δανεισμένα, ο φοιτητής μπορεί να κάνει **κράτηση (place a reservation).**
- Η κράτηση(reservation) καταχωρίζεται στον τίτλο του βιβλίου. Ένας φοιτητής μπορεί επίσης να **επεκτείνει τον δανεισμό του (extend their loan)**, εφόσον το βιβλίο δεν έχει ήδη κρατηθεί από άλλον.
- Όταν ένα αντίτυπο του συγκεκριμένου βιβλίου γίνει διαθέσιμο, ο **βιβλιοθηκάριος** το **δεσμεύει(a copy is reserved)** για τον φοιτητή και ο **φοιτητής ειδοποιείται(student is notified)** ότι το αντίτυπό του είναι έτοιμο για παραλαβή.

University online Library system
Sample solution



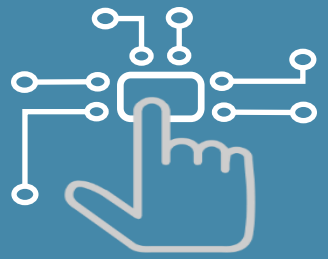


BREAKOUT SESSION

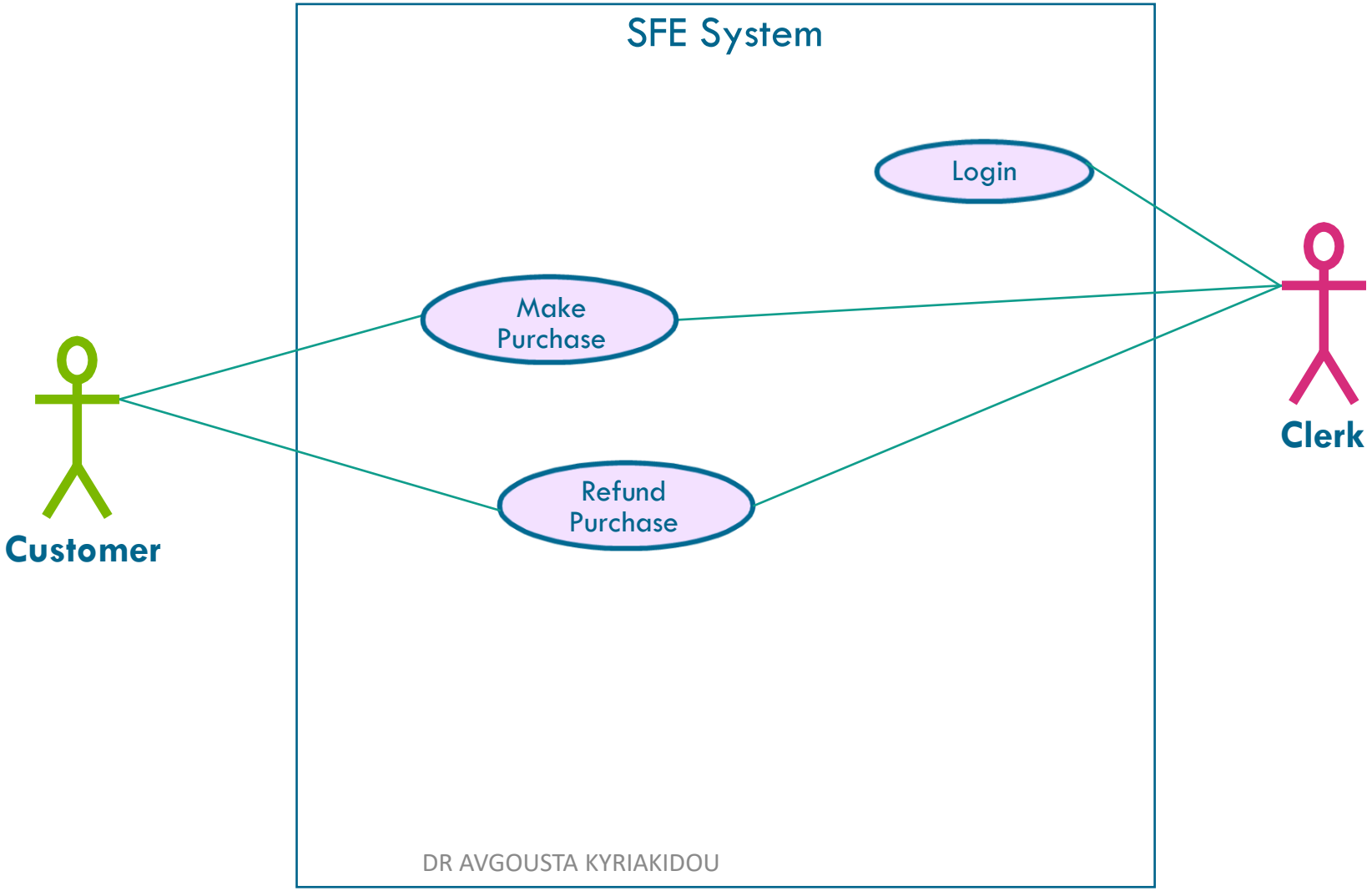


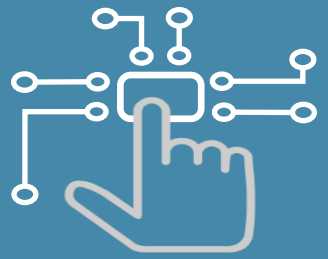
Exercise 2
Something for Everyone – SFE Case Study

- SFE is a shop which sells various products.
- **Customers** can purchase products from clerks.
- In order to **make a sale**, **clerks** scan items, calculate the total and obtain the payment from the customer.
- A **payment** can be done by cash or credit card.
- A customer can refund a purchase. For this to happen the clerk will ask for the receipt and will refund the payment (cash or credit).



Example – SFE


SFE System
Login
Make Purchase
Refund Purchase
Customer
Clerk

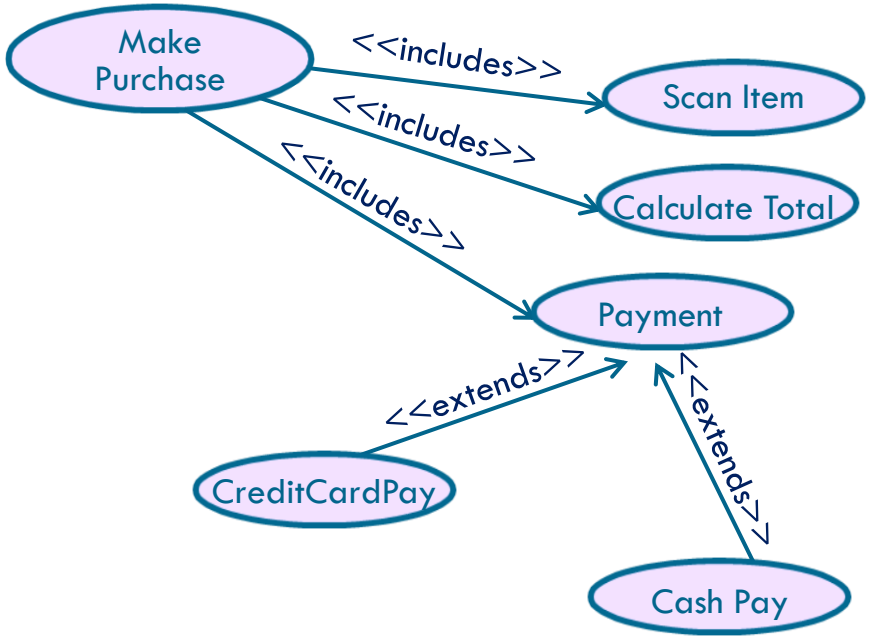


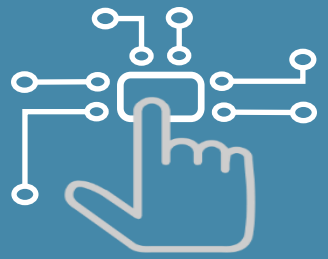
Example – SFE

SFE System

In order to **make a sale**, **clerks** scan items, calculate the total and obtain the payment from the customer.

A payment can be done by cash or credit card.





Example – SFE

SFE System

A **customer** can refund a purchase. For this to happen the **clerk** will ask for the receipt and will refund the payment (cash or credit).