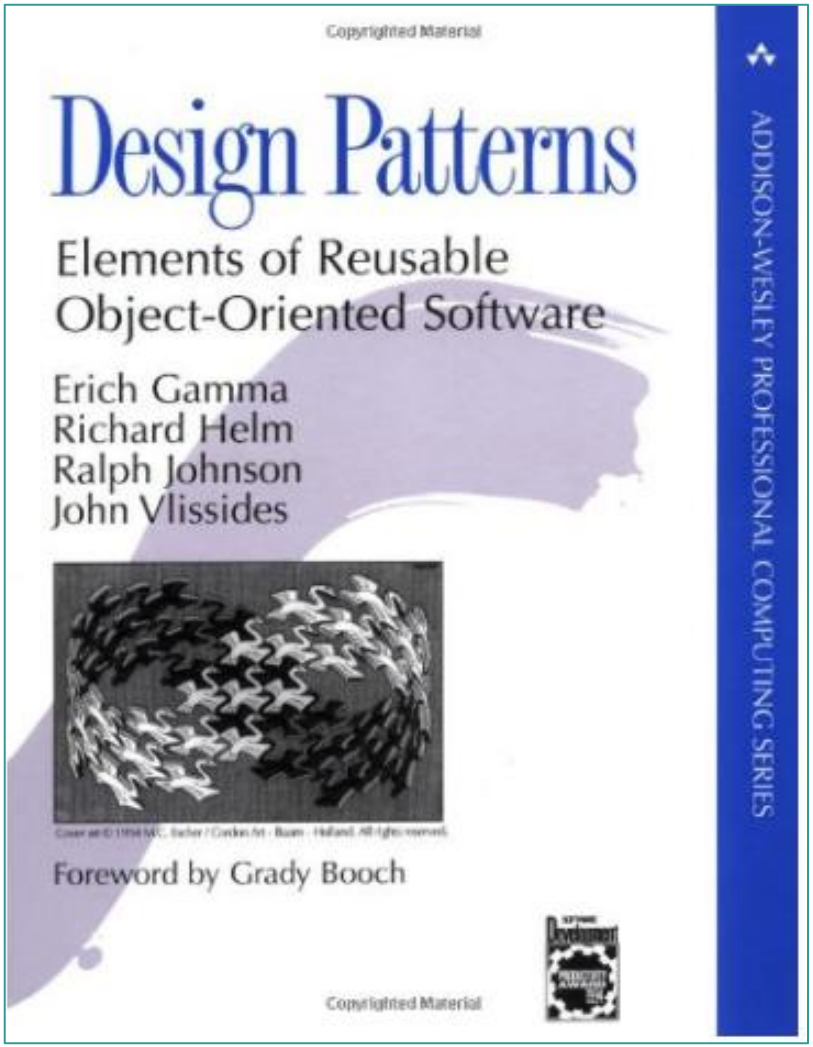
ΟΙΚΟΝΟΜΙΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ
ATHENS UNIVERSITY OF ECONOMICS AND BUSINESS
ΚΕΝΤΡΟ ΕΠΙΜΟΡΦΩΣΗΣ ΚΑΙ ΔΙΑ ΒΙΟΥ ΜΑΘΗΣΗΣ

Layered Architectures Client-Server, MVC, MVP DAO, DTOs, SOA

Αθ. Ανδρούτσος

Design Patterns - GoF



- Το 'Gang of Four' (GoF) Design Patterns είναι η συλλογή 23 design patterns από το βιβλίο "Design Patterns: Elements of Reusable Object-Oriented Software", 1994.
- Επηρέασε το software development
- Αυτά τα 23 Design Patterns επιτρέπουν στους προγραμματιστές να δημιουργήσουν επαναχρησιμοποιήσιμα προγράμματα χωρίς να χρειάζεται να ανακαλύψουν ξανά τις ίδιες τις σχεδιαστικές λύσεις. Περιλαμβάνουν:
- 5 creational patterns
- 7 structural patterns
- 11 behavioral patterns

Design Patterns - GoF

Pattern Category	Meaning in Life	Classic Patterns of this Category
Structural	These patterns are concerned with how classes and objects can be composed to obtain new functionality.	Adapter, Bridge, Composite, Decorator, Façade, Flyweight, Proxy
Creational	These patterns have to do with how objects are instantiated into memory, typically in an attempt to simplify their creation.	Abstract Factory, Builder, Factory Method, Prototype, Singleton
Behavioral	These patterns are concerned with communication between objects.	Chain of Responsibility, Command, Interpreter, Iterator, Mediator, Memento, Observer, State, Strategy, Template Method, Visitor

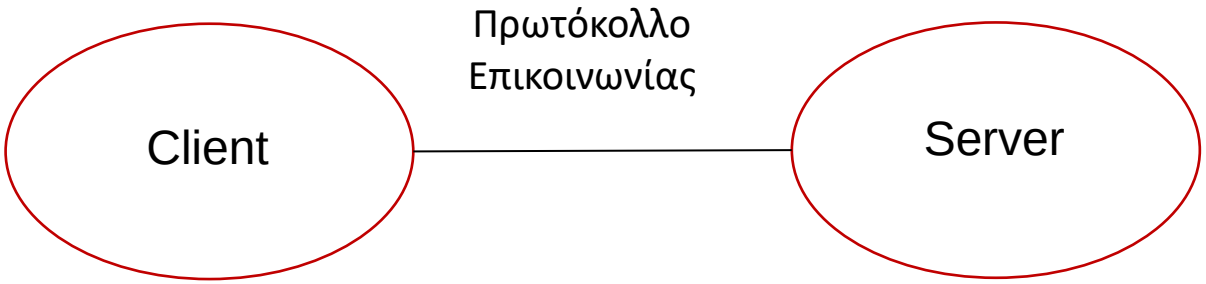
Java Enterprise Design Patterns

- Και η Java Enterprise Edition (τώρα πλέον Jakarta EE) έχει προτείνει Design Patterns για Enterprise εφαρμογές

Java Enterprise Design Patterns

- Delegation Pattern (Composition & Forwarding)
- Data Access Object (DAO)
- Data Transfer Object (DTO)
- Front Controller Pattern

Client - Server



- Ιστορικά το πρώτο και βασικό μοντέλο κατανεμημένης επικοινωνίας ήταν το μοντέλο **Client-Server (Πελάτης-Εξυπηρετητής)**.
- Ο **Client** (Πελάτης) είναι ένα πρόγραμμα που ζητάει / αιτείται (**requests**) υπηρεσίες από τον Server, τυπικά καλώντας μία μέθοδο του Server
- Ο **Server** (Εξυπηρετητής) είναι ένα πρόγραμμα που παρέχει υπηρεσίες μέσω ενός public API

Echo Server

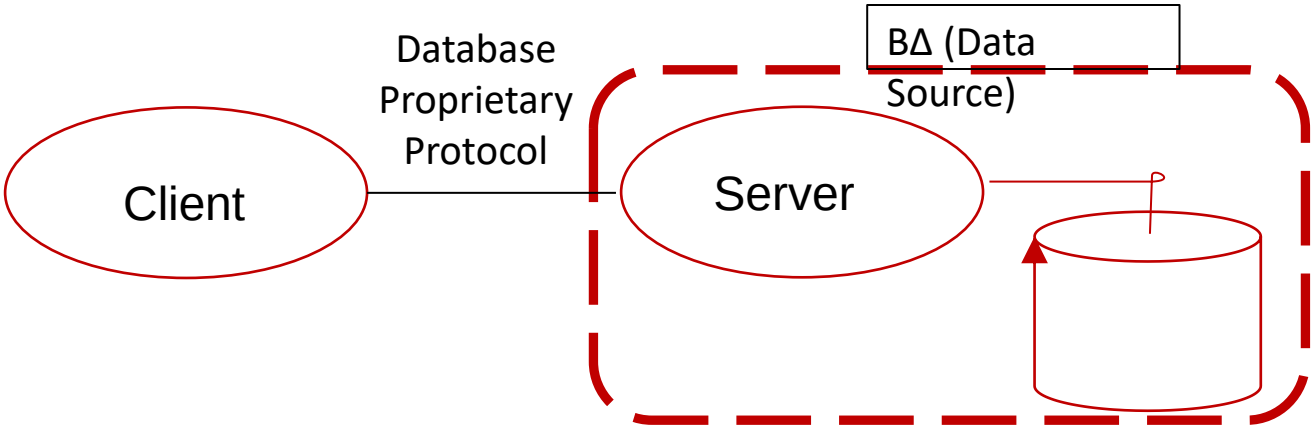

```java
package testbed.ch18.cliserv;

/**
 * A simple POJO class that acts as ech server.
 *
 * @author a8ana
 * @version 0.1
 */
public class EchoServ {

    /**
     * Echos a <code>String</code> carried by the <code>message</code>
     * argument.
     *
     * @param   message     string that contains the message to be echoed.
     * @return              the echoed message.
     */
    public String echo(String message) {
        return message;
    }
}
```

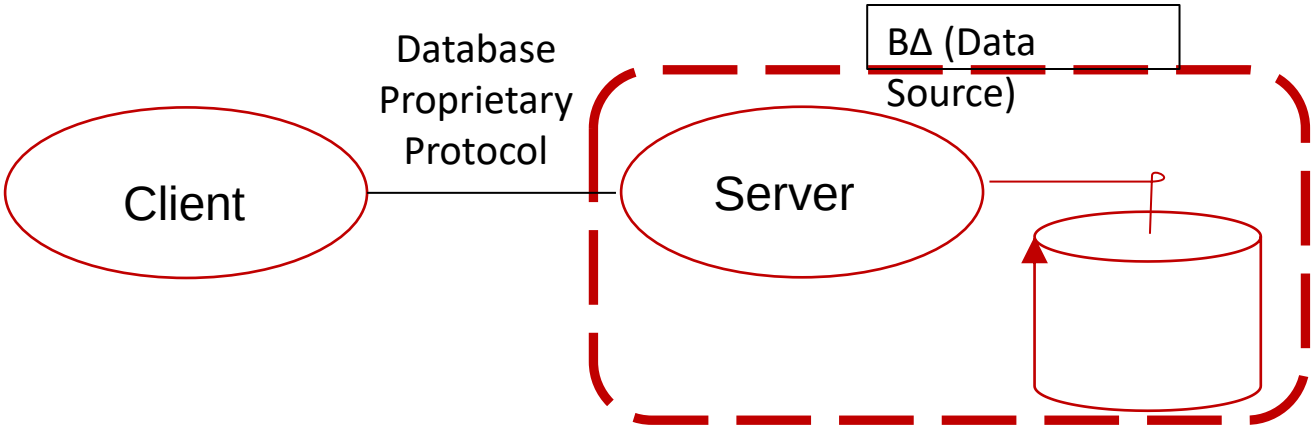
- Ο Echo Server επιστρέφει το message που δέχεται
- Αυτή είναι η υπηρεσία που παρέχει ο Echo Server
- Τεκμηριώνουμε κατάλληλα με Javadoc

Two-Tier Model (1)



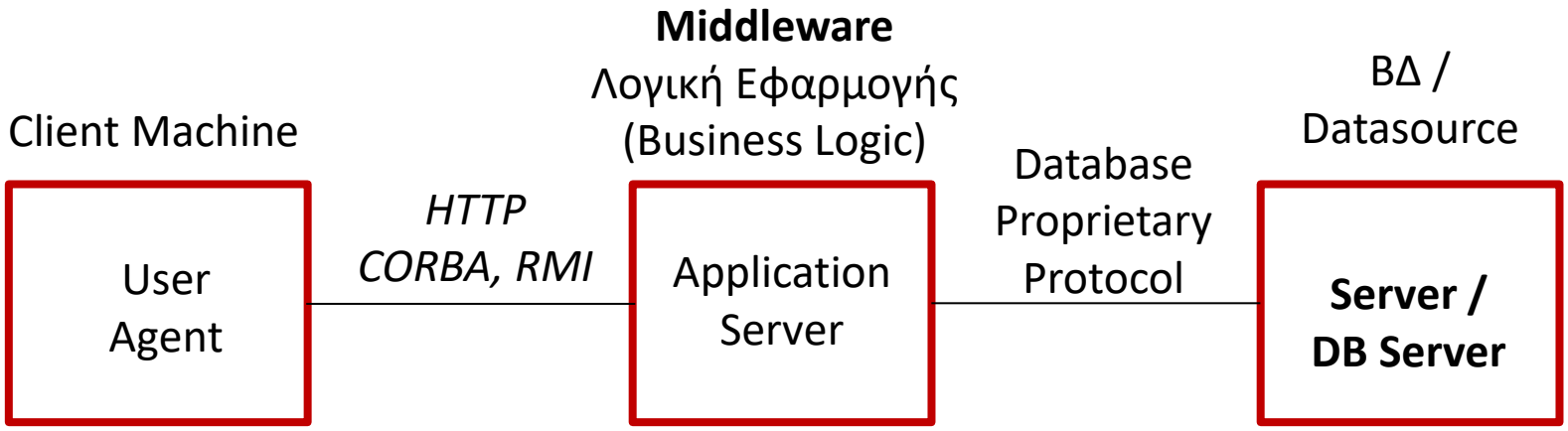
- Στη θέση του Server θα μπορούσε να είναι οποιαδήποτε υπηρεσία. Στην περίπτωση εμπορικών ή άλλων εφαρμογών που διαχειρίζονται δεδομένα, ο Server (Database Server) είναι ένα πρόγραμμα RDBMS όπως για παράδειγμα SQL Server, Oracle Server, MySQL Server, κλπ. που δέχεται requests (SQL commands) και επιστρέφει τα αποτελέσματα
- Το API του SQL Server είναι οι εντολές της γλώσσας SQL όπως παρέχονται προς τον client μέσω ενός πρωτοκόλλου επικοινωνίας καθώς και μέσω ενός προγράμματος (Driver) που κάνει τη μετάφραση μεταξύ του client (π.χ. Java) σε SQL

Two-Tier Model (2)



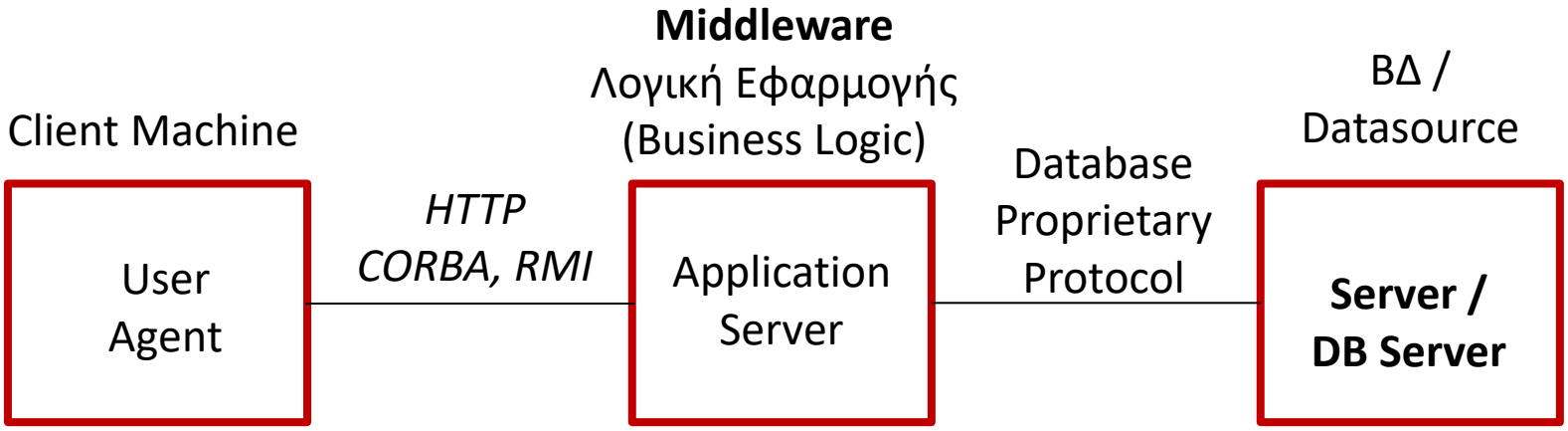
- Το μοντέλο δύο επιπέδων δεν είναι scalable γιατί η λογική της εφαρμογής βρίσκεται στον client και καθώς μεγαλώνει οδηγεί σε **Fat clients** όπου χρειαζόμαστε πολλά resources στο client machine (μνήμη, επεξεργαστή)

Three-Tier Model (1)



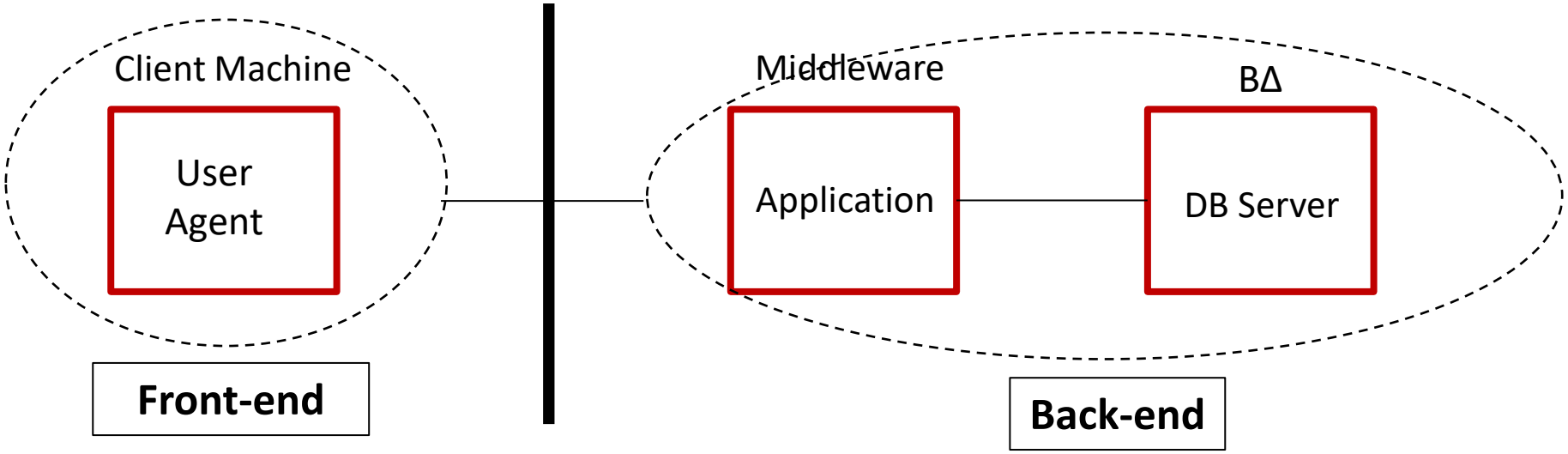
- Στο μοντέλο τριών επιπέδων (Three-Tier Architecture), η λογική της εφαρμογής φεύγει από τον client και πάει σε ένα ενδιάμεσο επίπεδο, το **Middleware**
- Αυτή η **αρχιτεκτονική τριών επιπέδων** είναι και το τυπικό setup μιας εμπορικής ή άλλης κατανεμημένης εφαρμογής
- Στο μοντέλο τριών επιπέδων οι αιτήσεις του client αποστέλλονται στο Middleware που υλοποιεί την επιχειρησιακή λογική και παρέχει τις αντίστοιχες υπηρεσίες

Three-Tier Model (2)



- Αποτελείται από τρία μέρη:
  - Τον **User Agent**, που μπορεί να είναι ένας χρήστης (π.χ. browser) ή όχι ένας χρήστης αλλά ένα πρόγραμμα
  - Το **Middleware** που υλοποιεί τη λογική (business logic) της εφαρμογής
  - Το **Datasource**, που τυπικά είναι μία ΒΔ και λειτουργεί ως το Persistence Layer της εφαρμογής, δηλαδή ως αποθήκη (storage) μόνιμης αποθήκευσης των δεδομένων

Front-end / Back-end

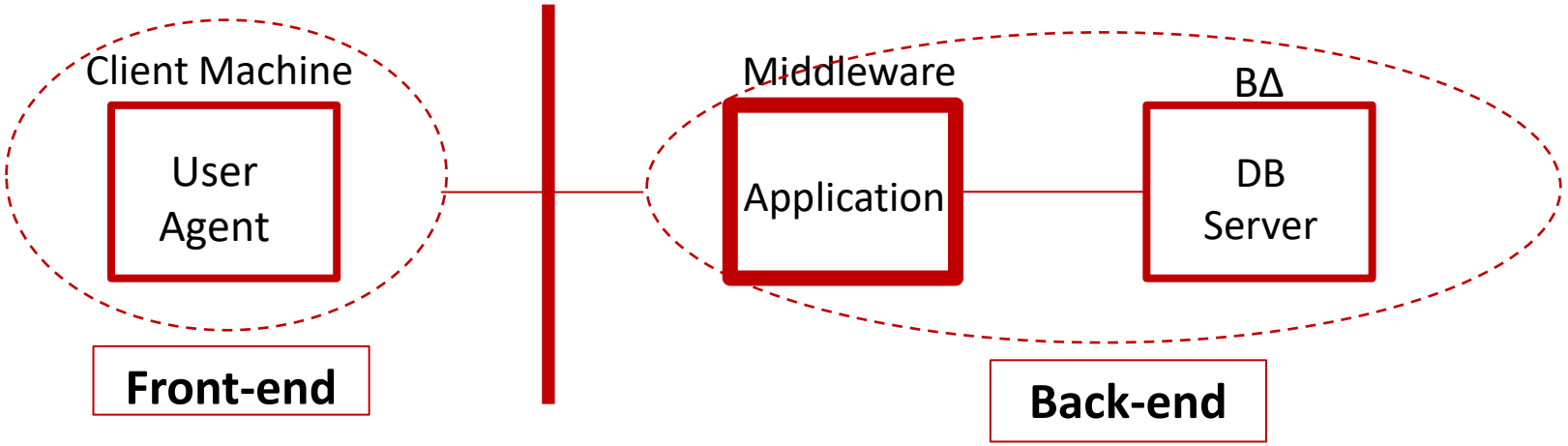


- Σε μία κατανεμημένη εφαρμογή υπάρχουν δύο άκρα (*-end), το Front-End και το Back-End
- Με τον όρο **Front-End** εννοούμε είναι τον User Agent (το μέρος του Client / Χρήστη)
- Το **Back-End** είναι το Middleware και η ΒΔ

Τεχνολογίες

- **Front-end** (Μπορεί να είναι το desktop σε μία desktop εφαρμογή ή ένα Graphical User Interface (GUI) σε μία GUI εφαρμογή – ή ο browser σε μία Web-based ή η φορητή συσκευή σε μία mobile app)**.**
- Οι τεχνολογίες ανά περίπτωση είναι οι παρακάτω:
  - ***Desktop Front-End:*** Java Swing, Java FX, .NET Windows Forms
  - ***Web Front-End****.* HTML/CSS/JavaScript
  - **Mobile.** Android, iOS
- **Back-end.**
  - *Java, C#, Python, JavaScript (Node.js), PHP, Ruby, κ.α.*

Three-Tier

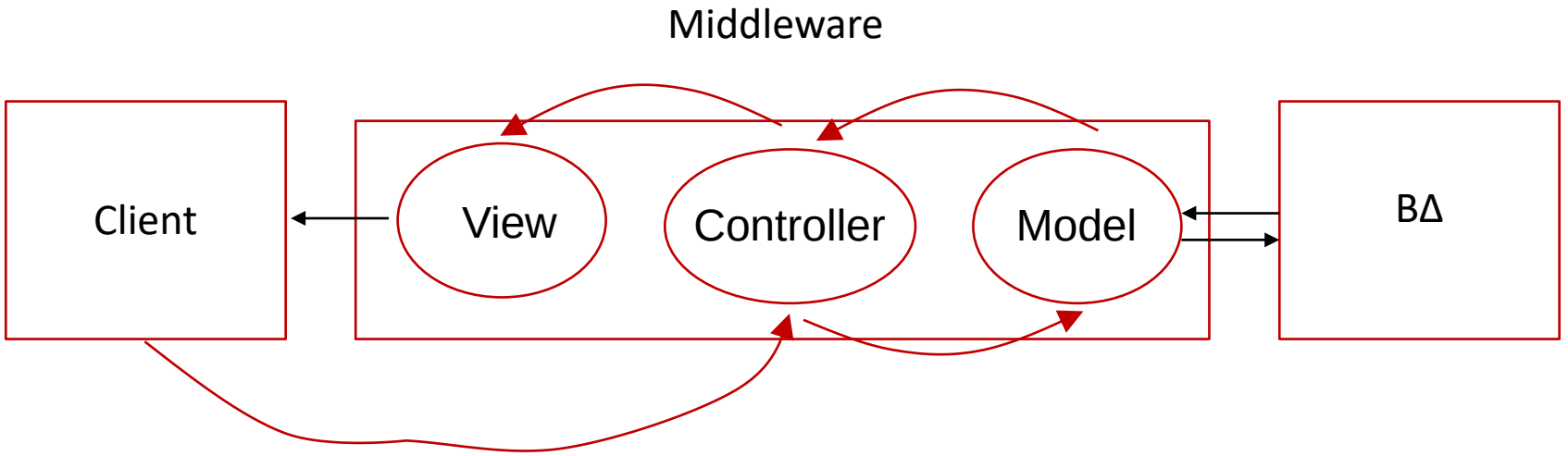


- Στην Αρχιτεκτονική τριών επιπέδων (Three-Tier) το **Middleware** είναι ένα σύνθετο component, που είναι υπεύθυνο για τα ακόλουθα concerns:
    1. **Αλληλεπιδρά** με τον client, λαμβάνει requests και επιστρέφει responses
    2. Δημιουργεί τα responses που μπορεί να είναι οθόνες (**Views**) προς τον client (π.χ. Web σελίδες) ή πληροφορίες (JSON/XML)
    3. Υλοποιεί 1) τα Δεδομένα της εφαρμογής, 2) τα Public APIs και 3) CRUD πράξεις προς τη ΒΔ

MVC (Model-View-Controller)

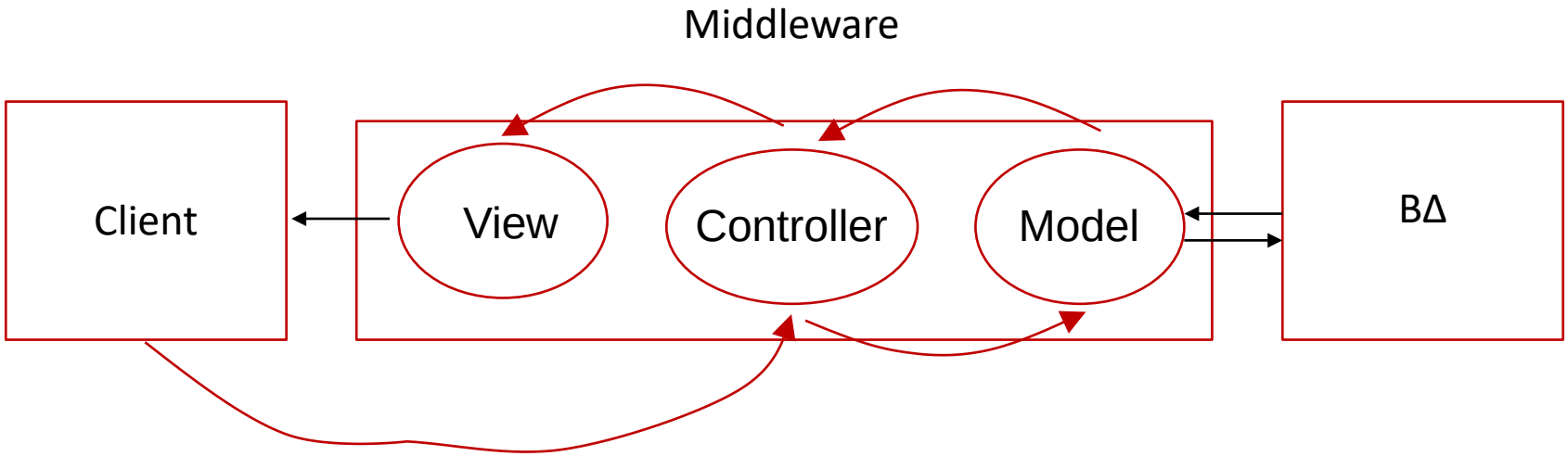
- Με βάση το Separation Of Concerns, το Middleware μπορεί περαιτέρω να διαχωριστεί σε τρία μέρη:
  - **Model**, που είναι μία αναπαράσταση των δεδομένων (Domain Model), του Public API (Services) και CRUD services
  - **View**, που αφορά τις οθόνες (views) που βλέπει ο χρήστης. Είναι μια οπτική αναπαράσταση των πληροφοριών προς τον χρήστη. Μπορούμε να δίνουμε διάφορα views στον χρήστη
  - **Controller**, που λαμβάνει τις αιτήσεις του χρήστη (requests), καλεί τις αντίστοιχες υπηρεσίες του Model και επιστρέφει τα views και γενικά τα responses. Πρόκειται για **ενδιάμεση οντότητα** που δρομολογεί (routing) τις αιτήσεις του χρήστη στην αντίστοιχη υπηρεσία (Public API)

MVC (1)



- Ο Controller λαμβάνει requests από τον client, στέλνει τα requests στο Model καλώντας το Public API του Model, λαμβάνει πίσω τα αποτελέσματα, τα οποία διαμορφώνει ως View το οποίο και στέλνει πίσω στον Client

MVC (2)

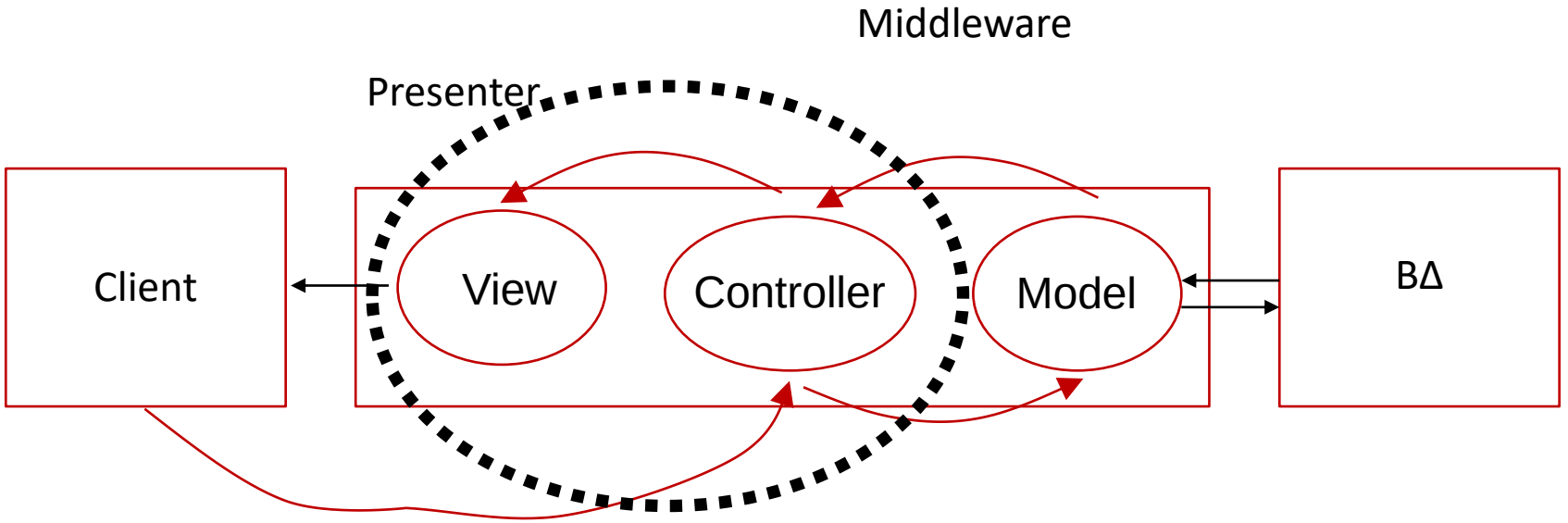


- Στη αρχιτεκτονική MVC το **Model** είναι ένα σύνθετο component, που είναι υπεύθυνο για τα ακόλουθα concerns:
    1. **Αλληλεπιδρά** με τον Controller (λαμβάνει requests από τον Controller και επιστρέφει αποτελέσματα)
    2. **Διαχειρίζεται** το Domain Model (Data classes) και **υλοποιεί** το Public API
    3. **Αλληλεπιδρά** με τον DB Server (πράξεις CRUD)

Παραλλαγές του MVC - MVP

- Στο MVC το View είναι passive δηλαδή δεν περιέχει λογική και τυπικά είναι γραμμένο σε XML ή HTML
- Στο μοντέλο MVP (Model-View-Presenter) το View δεν είναι passive αλλά περιέχει και μέρος της λογικής, τυπικά είναι γραμμένο σε μία γλώσσα όπως Java, C#, κ.α.
- MVVM (Model-View-ViewModel) – Εδώ το View επικοινωνεί απευθείας με το Model και αλληλοεπιδρούν άμεσα (Model-View Binding), δηλαδή όταν αλλάζει το Model, αλλάζει άμεσα και το View

MVP (Model-View-Presenter)



- Στο MVP, View και Controller δεν μπορούν να διαιρεθούν, όπως για παράδειγμα στην τεχνολογία Java Swing, γιατί το View γράφεται με Java classes και δεν μπορεί να είναι passive (όπως όταν γράφεται με HTML, XML)

MVC - Μονολιθική Προσέγγιση

- Το MVC είναι μία μονολιθική προσέγγιση γιατί το Model περιλαμβάνει διαφορετικά Concerns (**Data, Public API, CRUD**).
- Το παράδειγμα του Account όπου οι κλάσεις *Account*, *OverdraftAccount*, *JointAccount*, *OverdraftJointAccount* περιλαμβάναν η κάθε μία δεδομένα (**Data Model**), μεθόδους **CRUD**, καθώς και μεθόδους δημοσίων υπηρεσιών (**Public API**) είναι ένα παράδειγμα μονολιθικής προσέγγισης του Model Component

Μονολιθική Προσέγγιση

- Οι μονολιθικές προσεγγίσεις δεν είναι reusable γιατί δεν περιλαμβάνουν ανεξάρτητες μονάδες λογισμικού
- Περιλαμβάνουν μία μονάδα που κάνει τα πάντα (Data Model, Public API, CRUD), ούτε testable για τον ίδιο λόγο, δεν είναι εύκολο να συντηρηθούν (maintainable) ούτε να προσθέσουμε και άλλες υπηρεσίες (scalable)

Layered Architectures

- Θα μπορούσαμε να διασπάσουμε περαιτέρω το Model σε επιμέρους επίπεδα (Layers)
- Οι **στρωματοποιημένες αρχιτεκτονικές (Layered Architectures)** βασίζονται γενικά στη λογική της κατάτμησης του codebase σε μικρότερες ανεξάρτητες λογικές ενότητες (**Separation Of Concerns**) που είναι reusable, testable, maintainable, και scalable

Separation Of Concerns (SoC)

- Πρόκειται για *Design Principle* (σχεδιαστική αρχή) στην τεχνολογία λογισμικού σύμφωνα με την οποία το λογισμικό πρέπει να είναι περισσότερο επαναχρησιμοποιήσιμο (reusable), ελέγξιμο (testable) και παραγωγικότερο και για αυτό θα πρέπει να διαχωρίζεται σε ενότητες (layers) που αντιμετωπίζουν διαφορετικής φύσης προβλήματα (concerns)

CRUD (1)

- Ένα concern σε εφαρμογές που έχουν datasource είναι οι CRUD πράξεις που κάνει μία εφαρμογή, π.χ. προς μία ΒΔ και γενικά προς ένα persistence storage
- Στόχος είναι να κάνουμε τα data μας persist, δηλαδή να τα αποθηκεύσουμε με μόνιμο τρόπο και να μπορούμε στη συνέχεια να τα ανακτήσουμε

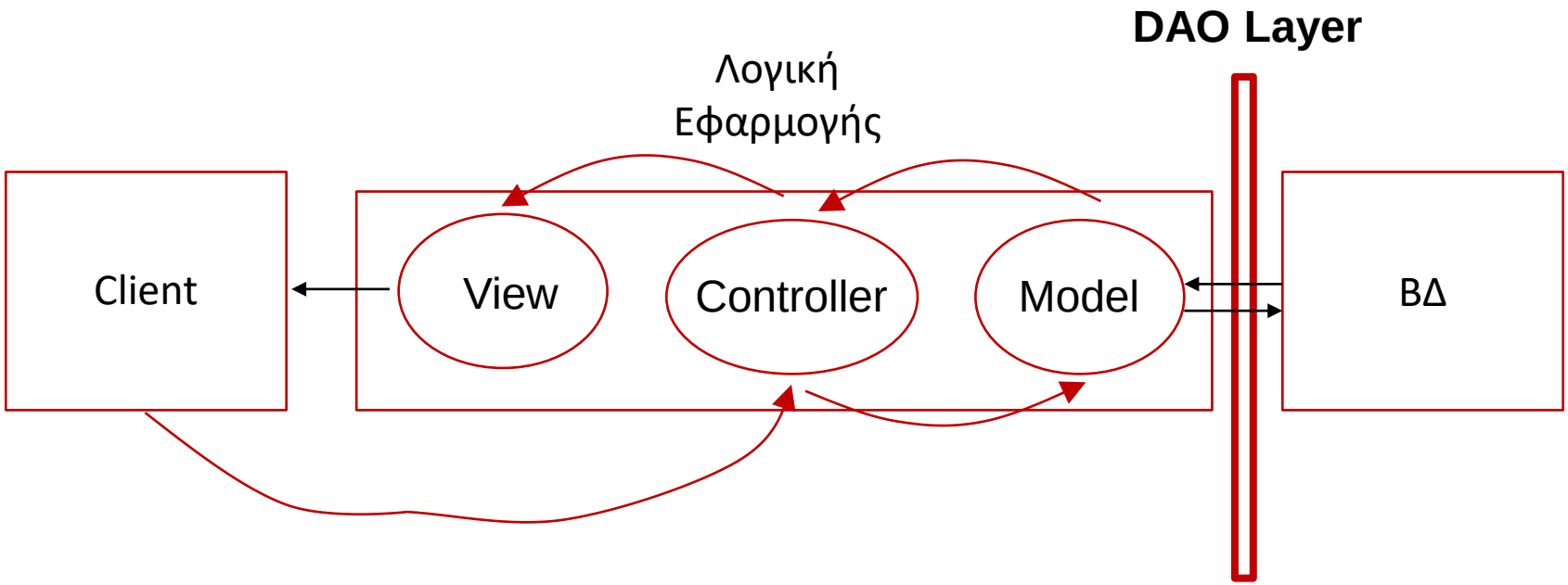
CRUD (2)

- Επειδή οι CRUD πράξεις είναι γραμμένες σε SQL και γενικά είναι γραμμένες στο API του storage, θα θέλαμε ιδανικά να αποκρύψουμε από την εφαρμογή μας αυτό το API και να παρέχουμε ένα προγραμματιστικό API (αν η εφαρμογή είναι Java, θα θέλαμε ένα Java API) που να παρέχει CRUD πράξεις όπως π.χ. add(), remove(), update(), getById() και όχι INSERT, DELETE, UPDATE, SELECT, αντίστοιχα

Data Access Objects (DAO)

- Το Data Access Object (DAO) είναι ένα Layer μεταξύ του Model και της ΒΔ και παρέχει προς την εφαρμογή μας CRUD υπηρεσίες μέσω ενός API που κάνει export (Public API)
- Επομένως η εφαρμογή μας δεν χρειάζεται να ασχολείται με το boilerplate των SQL CRUD μεθόδων αλλά μπορεί να λαμβάνει persistence services από το DAO

DAO Layer

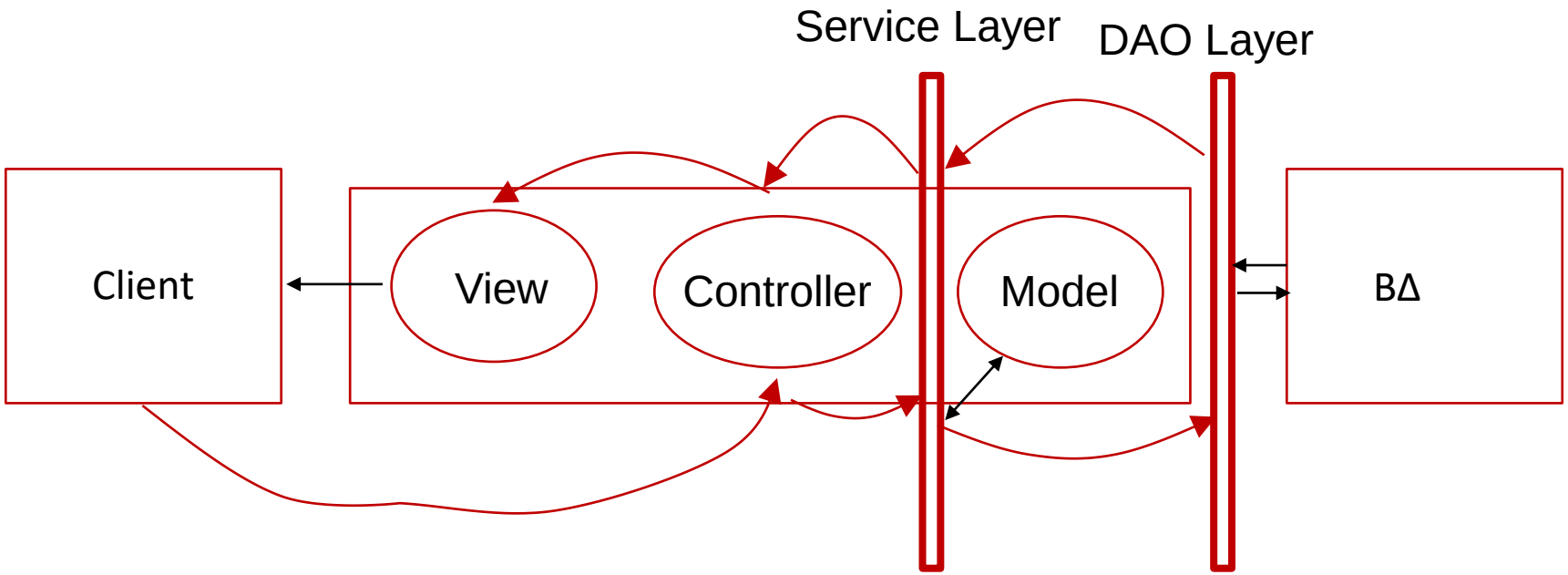


- Ανάμεσα στο Model και στη ΒΔ (γενικά datasource μιας και μπορεί να έχουμε και άλλες δομές, όπως in-memory DBs, Block Chain, Data Warehouses, κλπ.) εισέρχεται το DAO Layer, που παρέχει persistence services με τη μορφή ενός Public API

Service Layer

- Το πιο σημαντικό concern μίας εφαρμογής είναι οι υπηρεσίες που παρέχει η εφαρμογή προς τους clients, τυπικά το Public API
- Ιδανικά, θα θέλαμε αυτές οι υπηρεσίες να παρέχονται από ένα ανεξάρτητο Layer, το Service Layer

Service Layer

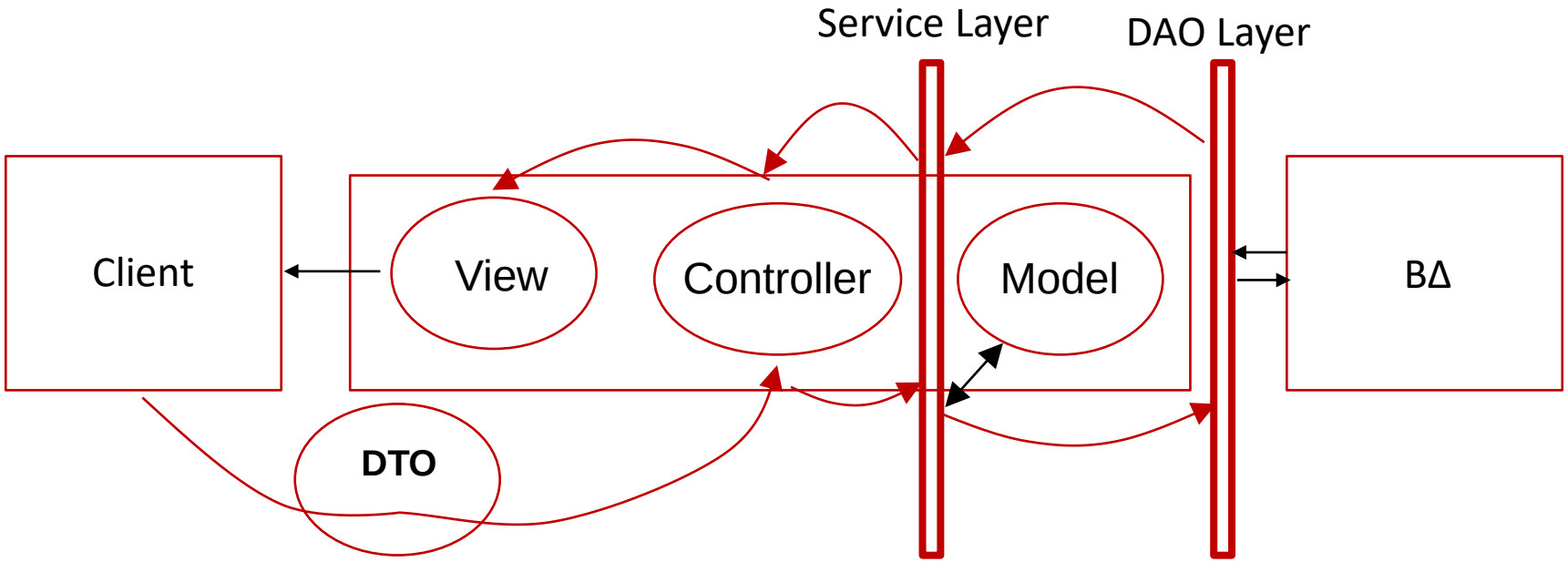


- Το service layer είναι ανεξάρτητο και αποτελείται από επιχειρησιακές υπηρεσίες που παρέχει η εφαρμογή μας στους clients, δηλαδή μόνο το Public API

Data Transfer Objects (DTO)

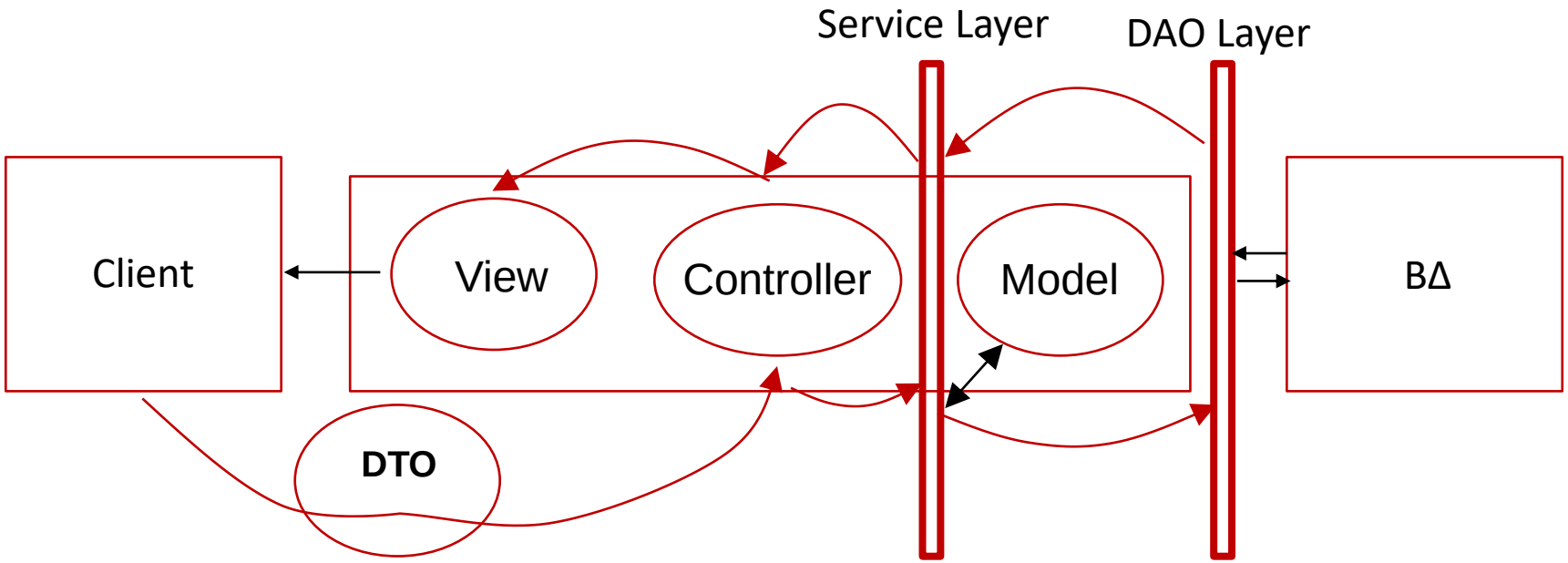
- Ένα ακόμα concern είναι η αλληλεπίδραση της εφαρμογής μας με τους clients. Ένας client μπορεί να αποστέλλει δεδομένα όπως τα στοιχεία μίας φόρμας ή να λαμβάνει δεδομένα, όπως τα αποτελέσματα μίας υπηρεσίας.
- Για τη μεταφορά των δεδομένων και προς τις δύο πλευρές μπορούμε να χρησιμοποιούμε 'πακέτα' πληροφοριών, που τυπικά είναι στην πλευρά του back-end είναι instances κλάσεων που ενθυλακώνουν αυτά τα δεδομένα.
- Αυτά τα object instances ονομάζονται Data Transfer Objects (DTO). Πρόκειται στην πραγματικότητα για το user model

DTOs



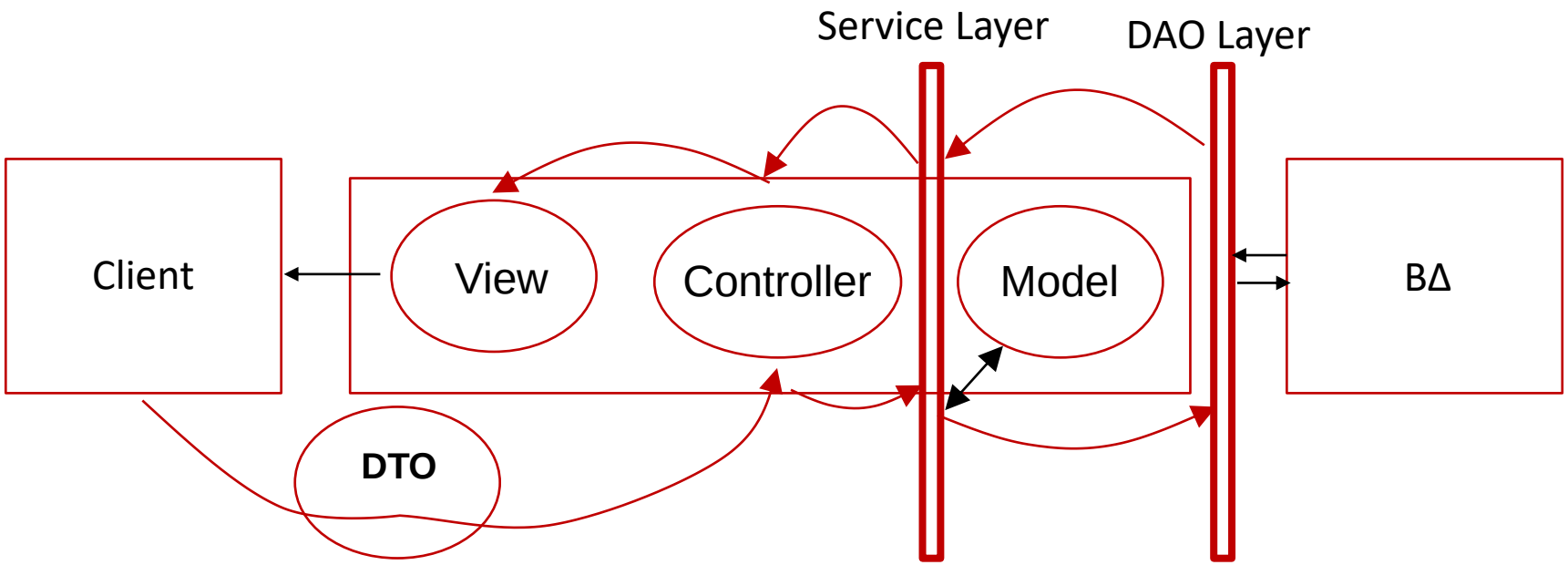
- Τα Data Transfer Objects (DTOs) είναι κλάσεις που χρησιμεύουν στη δημιουργία αντικειμένων μεταφοράς πληροφορίας από τον client στον Controller και από εκεί στο Service Layer και το αντίστροφο

N-Tier / SOA Architecture (1)



- Η αρχιτεκτονική που αναλύσαμε είναι μία N-Tier αρχιτεκτονική και συγκεκριμένα μία Service Oriented Architecture (SOA)

N-Tier / SOA Architecture (1)

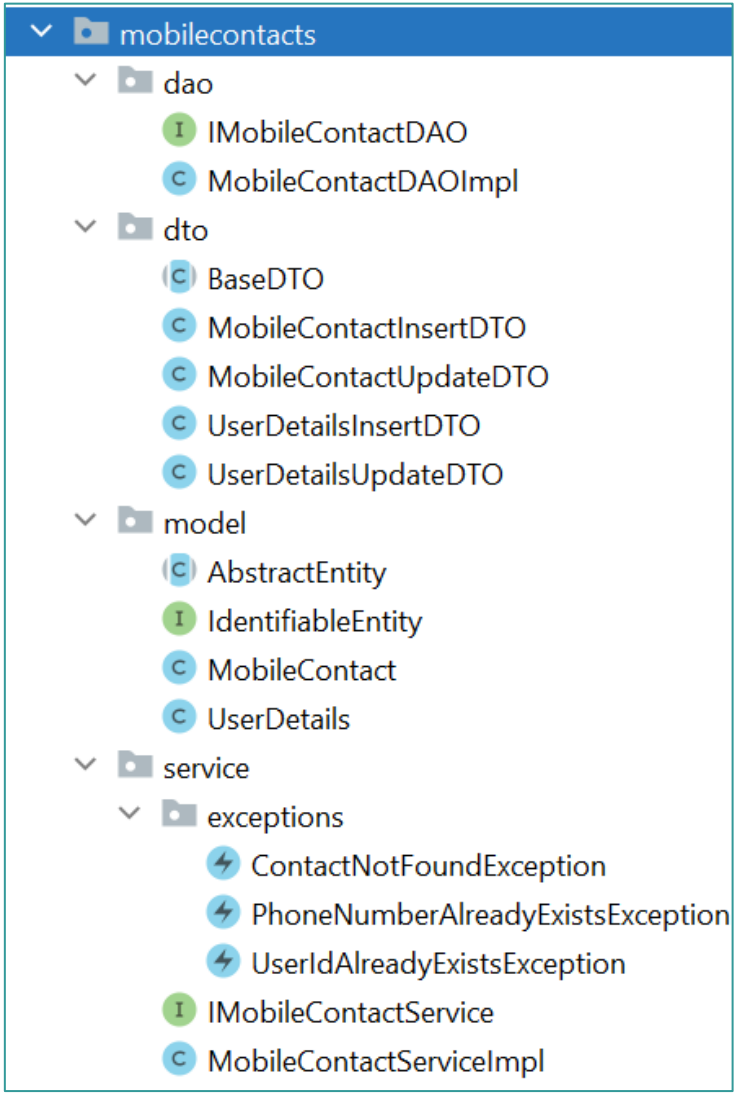


- Στην SOA το Model είναι απλά Data Classes (JavaBeans) που αναπαριστούν με abstract τρόπο τις οντότητες/πίνακες της ΒΔ

Δομή Project (1)

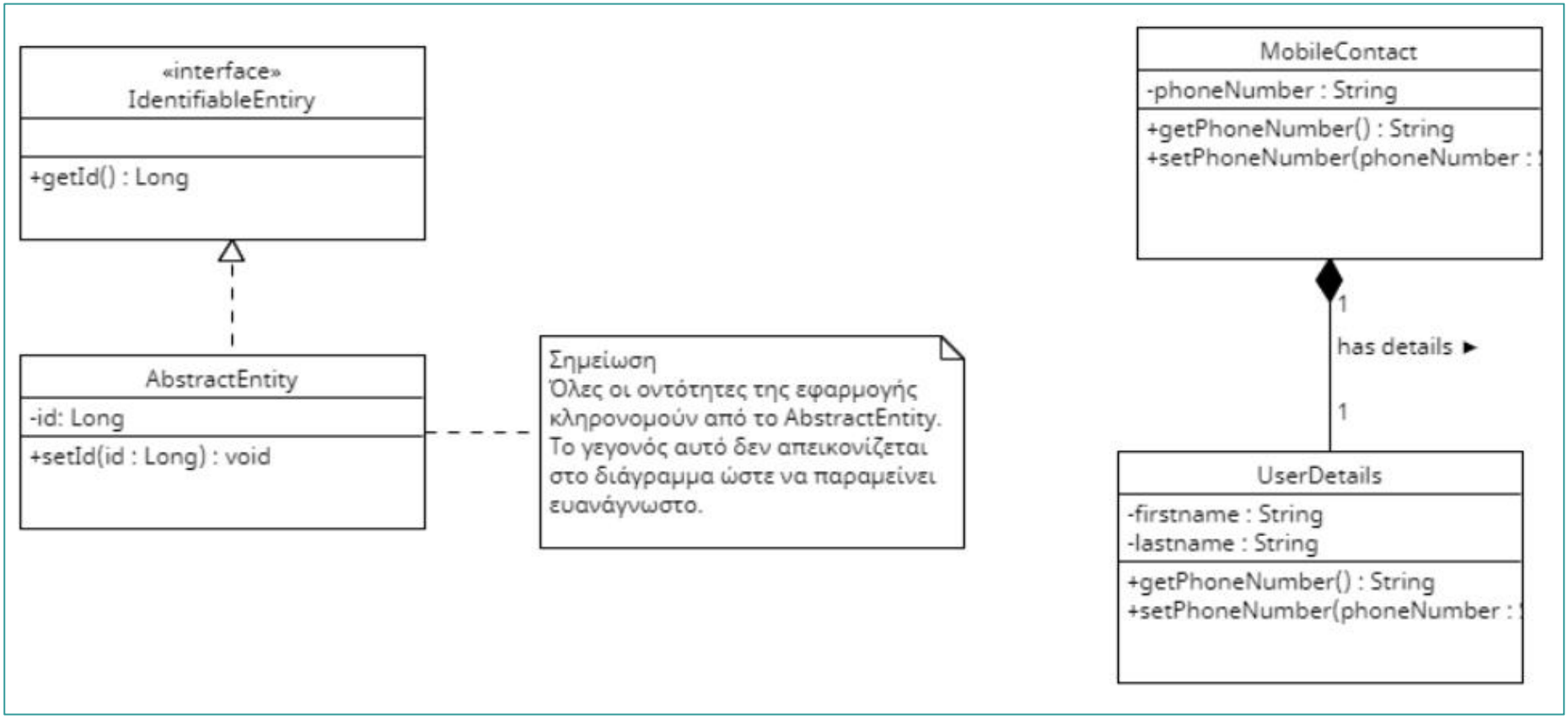
- Η δομή ενός project (οργάνωση φακέλων, υποφακέλων, αρχείων) είναι εξίσου σημαντικό χαρακτηριστικό του ποιοτικού λογισμικού, όσο το Code Style και το API Design
- Στο επόμενο παράδειγμα που αφορά τη Διαχείριση Mobile Contacts θα ακολουθήσουμε την αρχιτεκτονική SOA

Δομή Project (2)



- Η δομή του project φαίνεται στο αριστερό σχήμα
- Ακολουθεί τη λογική του Service Oriented Architecture με DAO και DTOs
- Κάθε Layer βρίσκεται σε ξεχωριστό package

Class Diagram



Interface Identifiable Entity


```
1   package testbed.ch18.mobilecontacts.model;
2
3   public interface IdentifiableEntity {
        1 implementation
4       Long getId();
5   }
```

- Θα οργανώσουμε το **model** μας σε μία ιεραρχία κλάσεων.
- Το βασικό κοινό πεδίο όλων των κλάσεων είναι το id. Το Public API που θα θέλαμε να δίνουν όλες οι κλάσεις του domain model είναι το getId(), να είναι identifiable

AbstractEntity


```java
package testbed.ch18.mobilecontacts.model;

public class AbstractEntity implements IdentifiableEntity {
    private Long id;

    @Override
    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }
}
```

- Το AbstractEntity λειτουργεί ως Skeletal implementation. Κάνει implements το IdentifiableEntity, υλοποιεί την getId() και παρέχει το id καθώς και ένα setter

User Details class


```java
package testbed.ch18.mobilecontacts.model;

public class UserDetails extends AbstractEntity {
    private String firstname;
    private String lastname;

    public UserDetails() {}

    public String getFirstname() { return firstname; }
    public void setFirstname(String firstname) { this.firstname = firstname; }
    public String getLastname() { return lastname; }
    public void setLastname(String lastname) { this.lastname = lastname; }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;

        UserDetails that = (UserDetails) o;

        if (!getFirstname().equals(that.getFirstname())) return false;
        return getLastname().equals(that.getLastname());
    }

    @Override
    public int hashCode() {
        int result = getFirstname().hashCode();
        result = 31 * result + getLastname().hashCode();
        return result;
    }
}
```

- Η κλάση UserDetails είναι κλάση του Domain Model
- Κάνει extends το AbstractEntity, οπότε είναι IdentifiableEntity

MobileContact class


```java
package testbed.ch18.mobilecontacts.model;

public class MobileContact extends AbstractEntity {
    private UserDetails userDetails;
    private String phoneNumber;

    public MobileContact() {}

    public UserDetails getUserDetails() { return userDetails; }
    public void setUserDetails(UserDetails userDetails) { this.userDetails = userDetails; }
    public String getPhoneNumber() { return phoneNumber; }
    public void setPhoneNumber(String phoneNumber) { this.phoneNumber = phoneNumber; }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;

        MobileContact that = (MobileContact) o;

        if (!getUserDetails().equals(that.getUserDetails())) return false;
        return getPhoneNumber().equals(that.getPhoneNumber());
    }

    @Override
    public int hashCode() {
        int result = getUserDetails().hashCode();
        result = 31 * result + getPhoneNumber().hashCode();
        return result;
    }
}
```

- Η MobileContact κάνει extends το AbstractEntity
- Με composition περιέχει ένα *User* private instance

DAO Layer (1)

- Το DAO Layer είναι ένα ενδιάμεσο επίπεδο (Proxy) μεταξύ του Datasource και οντοτήτων που αιτούνται υπηρεσιών CRUD
- Το Datasource κάνει persist τα data. Ωστόσο στο 1ο αυτό παράδειγμα θα θεωρήσουμε ένα in-memory datasource όπως ένα ArrayList
- Πρώτα θα ξεκινήσουμε και θα ορίσουμε το Public API του DAO Layer και στη συνέχεια θα το υλοποιήσουμε

DAO Layer (2)

- Τυπικά δημιουργούμε ένα νέο package με όνομα **dao,** ένα interface με όνομα *IMobileContactDAO* και μία κλάση με όνομα *MobileContactDAOImpl*
- Κατά σύμβαση τα interfaces που ορίζουν DAO APIs τελειώνουν σε DAO (π.χ. *IMobileContactDAO*) ενώ οι κλάσεις που τα υλοποιούν τελειώνουν σε DAOImpl (π.χ. *MobileContactDAOImpl*)

DAO - IMobileContactDAO


```java
package gr.aueb.cf.ch18.mobilecontacts.dao;

import gr.aueb.cf.ch18.mobilecontacts.model.MobileContact;
import java.util.List;

public interface IMobileContactDAO {

    /**
     * Inserts a new {@link MobileContact} instance in the datasource.
     *
     * @param mobileContact
     *          the model object that contains the mobile contact data.
     * @return
     *          the resulting {@link MobileContact}
     */
    MobileContact insert(MobileContact mobileContact);
```

- Ακολουθεί το interface IMobileContactDAO με doc comments. Η insert κάνει 'persist' το instance εισόδου

Update - Delete


```
18      /**
19       * Updates a  {@link MobileContact} instance.
20       *
21       * @param mobileContact
22       *          the model object that contains the mobile contact data.
23       * @return
24       *          the {@link MobileContact} instance before the update.
25       */
26      MobileContact update(MobileContact mobileContact);
27
28      /**
29       * Removes a {@link MobileContact} based on its id.
30       * @param id
31       *          the {@link MobileContact#id} of the instance
32       *          to be removed.
33       */
34      void delete(Long id);
```

- Η update είναι παρόμοια με την insert. Η delete διαγράφει με βάση το id

Get -GetAll


```
36      /**
37       * Returns a {@link MobileContact} based on the input id.
38       *
39       * @param id
40       *          the {@link MobileContact#id} of the instance
41       *          to be returnded.
42       * @return
43       *          the resulting {@link MobileContact}
44       */
45      MobileContact get(Long id);
46
47      /**
48       * Returns all the {@link MobileContact} instances from the
49       * datasource.
50       *
51       * @return
52       *          the resulting {@link List<MobileContact>}
53       */
54      List<MobileContact> getAll();
```

- Υλοποιούν τη λογική του SELECT (σε όρους ΒΔ)
- Η getAll υλοποιεί τη λογική του SELECT * FROM <MobileContacts>

Επιπλέον των βασικών CRUD


```
56      /**
57       * Returns a {@link MobileContact} based on the input phone-number.
58       *
59       * @param phoneNumber
60       *          the {@link MobileContact#phoneNumber} of the instance
61       *          to be returnded.
62       * @return
63       *          the resulting {@link MobileContact}
64       */
65      MobileContact get(String phoneNumber);
66
67      /**
68       * Checks if a phone number already exists as part
69       * of a {@link MobileContact} .
70       *
71       * @param phoneNumber
72       *          the <code>phoneNumber</code> to be searched for.
73       * @return
74       *          true, if <code>phoneNumber</code> exists in
75       *          a {@link MobileContact} in the datasource.
76       */
77      boolean phoneNumberExists(String phoneNumber);
```

- Επιπλέον των βασικών μεθόδων CRUD, μπορούμε να έχουμε μεθόδους όπως η get με βάση τον phone number που είναι unique, ή η phoneNumberExists, που ελέγχει αν υπάρχει ήδη ο phone number

Javadoc (5)


```
79      /**
80       * Checks if an <code>id</code> already exists as part
81       * of a {@link MobileContact}
82       *
83       * @param id
84       *          the <code>id</code> to be searched for.
85       * @return
86       *          true, if <code>id</code> exists in
87       *          a {@link MobileContact} in the datasource.
88       */
89      boolean userIdExists(Long id);
90
91      /**
92       * Removes a {@link MobileContact} instance from datasource.
93       *
94       * @param phoneNumber
95       *          the {@link MobileContact#phoneNumber} of the instance
96       *          needed to be removed.
97       */
98      void delete(String phoneNumber);
99  }
```

- Επίσης, παρέχουμε και άλλες επιπλέον CRUD, όπως userIdExists και delete με βάση το phoneNumber

DAO Implementation (1)


```
1   package gr.aueb.cf.ch18.mobilecontacts.dao;
2
3   import ...
8
9   public class MobileContactDAOImpl implements IMobileContactDAO {
10      private static final List<MobileContact> contacts = new ArrayList<>();
```

- Θεωρούμε ως Datasource ένα ArrayList. Πάνω σε αυτό το Datasource υλοποιούμε τις πράξεις CRUD που έχουμε ορίσει στο interface. Κατά σύμβαση το όνομα της κλάσης είναι το όνομα της Domain κλάσης και καταλήγει σε *Impl*

DAO Implementation (2)


```java
@Override
public MobileContact insert(MobileContact mobileContact) {
    contacts.add(mobileContact);
    return mobileContact;
}

@Override
public MobileContact update(MobileContact mobileContact) {
    return contacts.set(contacts.indexOf(mobileContact), mobileContact);
}

@Override
public void delete(Long id) {
    contacts.removeIf((contact) -> contact.getId().equals(id));
}

@Override
public MobileContact get(Long id) {
    Optional<MobileContact> contact = contacts.stream()
            .filter(c -> c.getId().equals(id))
            .findFirst();

    return contact.orElse(null);
}
```

- Με add, set κάνουμε insert και update την ArrayList
- Με .removeIf() που παρέχεται από την Collections class κάνουμε remove
- Για αναζητήσεις σε Collections χρησιμοποιούμε το Stream API
- Τα δεδομένα εισόδου θεωρούμε πως έχουν γίνει validate από το Service Layer

DAO Implementation (3)


```
38      @Override
39      public void delete(String phoneNumber) {
40          contacts.removeIf((contact) -> contact.getPhoneNumber().equals(phoneNumber));
41      }
42
43      @Override
44      public MobileContact get(String phoneNumber) {
45          Optional<MobileContact> contact = contacts.stream()
46                  .filter(c -> c.getPhoneNumber().equals(phoneNumber))
47                  .findFirst();
48
49          return contact.orElse(null);
50      }
51
52      @Override
53      public List<MobileContact> getAll() {
54          return new ArrayList<>(contacts);
55      }
```

- Υλοποιούμε και τις υπόλοιπες CRUD πράξεις

DAO Implementation (4)


```
57      @Override
58      public boolean phoneNumberExists(String phoneNumber) {
59          return contacts.stream()
60                  .anyMatch(c -> c.getPhoneNumber().equals(phoneNumber));
61      }
62
63      @Override
64      public boolean userIdExists(Long id) {
65          return contacts.stream()
66                  .anyMatch(c -> c.getId().equals(id));
67      }
68  }
```

- Τις αναζητήσεις σε Collections, όπως αναφέραμε και πιο πριν, τις κάνουμε με χρήση του Stream API

Data Transfer Objects (1)

- Τα Data Transfer Objects υλοποιούν το User Model και λειτουργούν ως οχήματα μεταφοράς δεδομένων
- Αν υποθέσουμε ότι ο χρήστης εισάγει πληροφορίες σε μία φόρμα Εισαγωγής μίας επαφής στο κινητό, τότε τα πεδία της φόρμας μπορούν να αναπαρασταθούν με ένα αντίστοιχο DTO

Data Transfer Objects (2)

- Επομένως με τα DTOs κάνουμε το binding μεταξύ των δεδομένων που βλέπει και εισάγει ο χρήστης στο Front-End και του επιπέδου που τα λαμβάνει στο Back-End όπως είναι το Service Layer, το οποίο λαμβάνει ως είσοδο DTOs
- Τυπικά δημιουργούμε ένα νέο package με όνομα **dto** και μέσα σε αυτό κλάσεις με ονόματα που κατά σύμβαση τελειώνουν σε DTO (π.χ. *MobileContactDTO*)

BaseDTO abstract class


```java
package gr.aueb.cf.ch18.mobilecontacts.dto;

public abstract class BaseDTO {
    private Long id;

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }
}
```

- Για λόγους δόμησης της λογικής των DTO και επαναχρησιμοποίησης του κώδικα και ευελιξίας μπορούμε να έχουμε ένα BaseDTO που να έχει το κοινό στοιχείο σε κάποια DTOs, που είναι το id

UserDetailsInsertDTO


```
1   package gr.aueb.cf.ch18.mobilecontacts.dto;
2
3   public class UserDetailsInsertDTO extends BaseDTO {
4       private String firstname;
5       private String lastname;
6
7       public UserDetailsInsertDTO() {}
8
9       public String getFirstname() { return firstname; }
12      public void setFirstname(String firstname) { this.firstname = firstname; }
15      public String getLastname() { return lastname; }
18      public void setLastname(String lastname) { this.lastname = lastname; }
21  }
```

- Πρόκειται για τα δεδομένα που δίνει ο χρήστης για τα User Details. Τα 'συσκευάζουμε' σε DTOs

UserDetailsUpdateDTO


```java
package gr.aueb.cf.ch18.mobilecontacts.dto;

public class UserDetailsUpdateDTO extends BaseDTO {
    private String firstname;
    private String lastname;

    public UserDetailsUpdateDTO() {}

    public String getFirstname() { return firstname; }
    public void setFirstname(String firstname) { this.firstname = firstname; }
    public String getLastname() { return lastname; }
    public void setLastname(String lastname) { this.lastname = lastname; }
}
```

MobileContactInsertDTO


```java
package gr.aueb.cf.ch18.mobilecontacts.dto;

public class MobileContactInsertDTO extends BaseDTO {
    private UserDetailsInsertDTO userDetailsInsertDto;
    private String phoneNumber;

    public MobileContactInsertDTO() {}
    public UserDetailsInsertDTO getUserDetailsDTO() { return userDetailsInsertDto; }
    public void setUserDetailsDTO(UserDetailsInsertDTO userDetailsInsertDTO) {
        this.userDetailsInsertDto = userDetailsInsertDTO;
    }
    public String getPhoneNumber() { return phoneNumber; }
    public void setPhoneNumber(String phoneNumber) { this.phoneNumber = phoneNumber; }
}
```

- Το MobileContactInsertDTO περιέχει και ένα UserDetailsInsertDTO

MobileContactUpdateDTO


```java
package gr.aueb.cf.ch18.mobilecontacts.dto;

public class MobileContactUpdateDTO extends BaseDTO {
    private UserDetailsUpdateDTO userDetailsUpdateDto;
    private String phoneNumber;

    public MobileContactUpdateDTO() {}

    public UserDetailsUpdateDTO getUserDetailsUpdateDto() { return userDetailsUpdateDto; }
    public void setUserDetailsUpdateDto(UserDetailsUpdateDTO userDetailsUpdateDto) {...}
    public String getPhoneNumber() { return phoneNumber; }
    public void setPhoneNumber(String phoneNumber) { this.phoneNumber = phoneNumber; }
}
```

- Το MobileContactUpdateDTO περιέχει και ένα UserDetailsUpdateDTO

Exceptions (1)

- Τα custom Exceptions είναι μέρος του Service Layer και υλοποιούν όλη τη λογική των λογικών σφαλμάτων που μπορεί να συμβούν κατά την παροχή μίας υπηρεσίας
- Για παράδειγμα αν αναζητούμε ένα προϊόν, μία συνθήκη λάθους θα ήταν να μην υπάρχει το προϊόν
- Στην εφαρμογή μας τα custom exceptions που θεωρούμε είναι να υπάρχει ήδη το phone-number ενός Mobile Contact που αναζητούμε, να υπάρχει ήδη το id ενός Mobile Contact καθώς και να μην υπάρχει το Mobile Contact

Exceptions (2)

- Τυπικά δημιουργούμε ένα νέο package με όνομα **exceptions** και μέσα σε αυτό κλάσεις με ονόματα που κατά σύμβαση τελειώνουν σε Exception (π.χ. *MobileContactNotFoundException*)

Phone number already exists


```java
package gr.aueb.cf.ch18.mobilecontacts.service.exceptions;

import gr.aueb.cf.ch18.mobilecontacts.model.MobileContact;

public class PhoneNumberAlreadyExistsException extends Exception {
    private final static long serialVersionUID = 1L;

    public PhoneNumberAlreadyExistsException(MobileContact mobileContact) {
        super("Mobile contact with phone number " + mobileContact.getPhoneNumber() + " already exists");
    }
}
```

- Exception που δημιουργούμε όταν δεν βρεθεί ένας αριθμός τηλεφώνου

User id already exists


```java
package gr.aueb.cf.ch18.mobilecontacts.service.exceptions;

import gr.aueb.cf.ch18.mobilecontacts.model.MobileContact;

public class UserIdAlreadyExistsException extends Exception {

    private final static long serialVersionUID = 1L;

    public UserIdAlreadyExistsException(MobileContact mobileContact) {
        super("Mobile contact with id " + mobileContact.getId() + " already exists");
    }
}
```

- Exception που δημιουργούμε όταν δεν βρεθεί ένα id χρήστη

Contact not found exception


```java
package gr.aueb.cf.ch18.mobilecontacts.service.exceptions;

import gr.aueb.cf.ch18.mobilecontacts.model.MobileContact;

public class ContactNotFoundException extends Exception {
    private final static long serialVersionUID = 1L;

    public ContactNotFoundException(String phoneNumber) {
        super("The mobile contact with phone number " + phoneNumber + " was not found");
    }

    public ContactNotFoundException(Long id) { super("The mobile contact with id " + id + " was not found"); }

    public ContactNotFoundException(MobileContact mobileContact) {
        super("The mobile contact with id " + mobileContact.getId()
                + " and phone number " + mobileContact.getPhoneNumber() + " was not found");
    }
}
```

- Exception που δημιουργούμε όταν δεν βρεθεί μία επαφή

Service Layer (1)

- Το Service Layer είναι το σημαντικότερο Layer όσο αφορά το σχεδιασμό του Public API
- Πρόκειται για τις υπηρεσίες που παρέχει η εφαρμογή μας προς τους clients
- Για να φέρει εις πέρας το Service Layer τις υπηρεσίες αυτές χρησιμοποιεί τις υπηρεσίες του DAO Layer (Composition & Forwarding, Delegation Pattern, Proxy Pattern, Decorator Pattern)

Service Layer (2)

- Τυπικά δημιουργούμε ένα νέο package με όνομα **service,** και interface με όνομα *IMobileContactService καθώς* και μία κλάση με όνομα *MobileContactServiceImpl*
- Κατά σύμβαση τα interfaces που ορίζουν Services τελειώνουν σε Service (π.χ. *IMobileContactService*) ενώ οι κλάσεις που τα υλοποιούν τελειώνουν σε ServiceImpl (π.χ. *MobileContactServiceImpl*)

Service Layer (3)

- Οι υπηρεσίες του Service Layer δέχονται DTOs, τα κάνουν map σε Model instances (μιας και το DAO θέλει Model instances) και καλούν DAO services
- Άλλες φορές αν τα DTOs περιέχουν IDs οντοτήτων θα πρέπει τα instances των οντοτήτων να γίνουν extract με κλήσεις του DAO προς την DB

Service Layer (4)

- Η λογική επομένως των υπηρεσιών στο Service Layer είναι:
  - Το Service λαμβάνει το DTO και κάνει map ή extract την πληροφορία και μετατρέπει τελικά σε Model instance. Για το σκοπό αυτό μπορεί να χρησιμοποιεί private methods που κάνουν αυτή τη δουλειά (map ή extract)
  - Μέσω του DAO instance (που έχει κάνει wiring) επιτελεί τα CRUD services χρησιμοποιώντας model instances
  - Επιστρέφει προς τον caller το service το οποίο υλοποιεί

IMobileContactService (1)


```java
package gr.aueb.cf.ch18.mobilecontacts.service;

import ...


public interface IMobileContactService {

    /**
     * Creates a {@link MobileContact} based on the data carried by the {@link MobileContactInsertDTO}
     *
     * @param mobileContactInsertDTO
     *          the DTO object that contains the mobile contact data.
     * @return
     *          the resulting {@link MobileContact}
     * @throws PhoneNumberAlreadyExistsException
     *          if the {@link MobileContact#phoneNumber} already exists in
     *          the datasource.
     * @throws UserIdAlreadyExistsException
     *          if {@link MobileContact#id} already exists in the datasource.
     */
    MobileContact insertMobileContact(MobileContactInsertDTO mobileContactInsertDTO)
            throws PhoneNumberAlreadyExistsException, UserIdAlreadyExistsException;
```

IMobileContactService (2)


```
32      /**
33       * Updates a {@link MobileContact} based on the data carried by the {@link MobileContactInsertDTO} .
34       *
35       * @param mobileContactUpdateDto
36       *          the DTO object that contains the mobile contact data.
37       * @return
38       *          the {@link MobileContact} before the update.
39       * @throws PhoneNumberAlreadyExistsException
40       *          if the new {@link MobileContact#phoneNumber} already exists in
41       *          the datasource.
42       * @throws ContactNotFoundException
43       *          if {@link MobileContactUpdateDTO#id} does not map
44       *          to a {@link MobileContact} in the datasource.
45       */
46      MobileContact updateContact(MobileContactUpdateDTO mobileContactUpdateDto)
47              throws PhoneNumberAlreadyExistsException, ContactNotFoundException;
```

IMobileContactService (3)


```
49      /**
50       * Removes a {@link MobileContact}.
51       *
52       * @param id
53       *          the {@link MobileContact#id} needed to be removed.
54       * @throws ContactNotFoundException
55       *          if {@link MobileContact#id} does not map
56       *          to a {@link MobileContact} in the datasource.
57       */
58      void deleteContactById(Long id) throws ContactNotFoundException;
59
60      /**
61       * Returns the {@link MobileContact} based on the input id.
62       *
63       * @param id
64       *          the {@link MobileContact#id} of the mobile contact instance
65       *          needed to be returned.
66       * @return
67       *          the resulting {@link MobileContact}
68       * @throws ContactNotFoundException
69       *          if {@link MobileContact#id} does not map
70       *          to a {@link MobileContact} in the datasource.
71       *
72       */
73      MobileContact getContact(Long id) throws ContactNotFoundException;
```

IMobileContactService (4)


```
75      /**
76       * Returns all the {@link MobileContact} instances of the datasource.
77       *
78       * @return
79       *      the resulting {@link List<MobileContact>.}
80       */
81      List<MobileContact> getAllContacts();
82
83      /**
84       * Returns the {@link MobileContact} based on the input phone number.
85       *
86       * @param phoneNumber
87       *          the {@link MobileContact#phoneNumber} of the mobile contact instance
88       *          needed to be returned.
89       * @return
90       *          the resulting {@link MobileContact}
91       * @throws ContactNotFoundException
92       *          if {@link MobileContact#phoneNumber} does not map
93       *          to a {@link MobileContact} in the datasource.
94       *
95       */
96      MobileContact getContact(String phoneNumber) throws ContactNotFoundException;
```

IMobileContactService (5)


```
96      MobileContact getContact(String phoneNumber) throws ContactNotFoundException;
97
98      /**
99       * Removes a {@link MobileContact}.
100      *
101      * @param phoneNumber
102      *          the {@link MobileContact#phoneNumber} needed to be removed.
103      * @throws ContactNotFoundException
104      *          if {@link MobileContact#phoneNumber} does not map
105      *          to a {@link MobileContact} in the datasource.
106      */
107     void deleteContact(String phoneNumber) throws ContactNotFoundException;
108 }
109
```

Service Layer Implementation (1)


```
1   package gr.aueb.cf.ch18.mobilecontacts.service;
2
3   import ...
15
16  public class MobileContactServiceImpl implements IMobileContactService {
17
18      // Composition
19      private final IMobileContactDAO dao;
20
21      // Wiring DAO to Service
22      public MobileContactServiceImpl(IMobileContactDAO dao) { this.dao = dao; }
```

- Ένα σημαντικό βήμα είναι το **wiring** του dao στο service layer (dependency injection)

Insert


```
26      @Override
27      public MobileContact insertMobileContact(MobileContactInsertDTO mobileContactInsertDto)
28              throws PhoneNumberAlreadyExistsException, UserIdAlreadyExistsException {
29          MobileContact mobileContact;
30          try {
31              mobileContact = new MobileContact();
32              mapInsertMobileContact(mobileContact, mobileContactInsertDto);
33
34              if (dao.phoneNumberExists(mobileContact.getPhoneNumber())) {
35                  throw new PhoneNumberAlreadyExistsException(mobileContact);
36              }
37              if (dao.userIdExists(mobileContact.getId())) {
38                  throw new UserIdAlreadyExistsException(mobileContact);
39              }
40
41              return dao.insert(mobileContact);
42          } catch (PhoneNumberAlreadyExistsException | UserIdAlreadyExistsException e) {
43              e.printStackTrace();
44              throw e;
45          }
46      }
```

Update


```
48      @Override
49      public MobileContact updateContact(MobileContactUpdateDTO mobileContactUpdateDto)
50              throws PhoneNumberAlreadyExistsException, ContactNotFoundException {
51          MobileContact mobileContact;
52
53          try {
54              mobileContact = new MobileContact();
55              mapUpdateMobileContact(mobileContact, mobileContactUpdateDto);
56
57              if (dao.phoneNumberExists(mobileContact.getPhoneNumber())) {
58                  throw new PhoneNumberAlreadyExistsException(mobileContact);
59              }
60              if (!dao.userIdExists(mobileContact.getId())) {
61                  throw new ContactNotFoundException(mobileContact);
62              }
63
64              return dao.update(mobileContact);
65          } catch (PhoneNumberAlreadyExistsException | ContactNotFoundException e) {
66              e.printStackTrace();
67              throw e;
68          }
69      }
```

Delete By id


```
71      @Override
72      public void deleteContactById(Long id) throws ContactNotFoundException {
73          try {
74              if (!dao.userIdExists(id)) {
75                  throw new ContactNotFoundException(id);
76              }
77
78              dao.delete(id);
79          } catch (ContactNotFoundException e) {
80              e.printStackTrace();
81              throw e;
82          }
83      }
```

getContact με βάση το id


```
85      @Override
86      public MobileContact getContact(Long id) throws ContactNotFoundException {
87          MobileContact mobileContact;
88          try {
89              mobileContact = dao.get(id);
90              if (mobileContact == null) {
91                  throw new ContactNotFoundException(id);
92              }
93              return mobileContact;
94          } catch (ContactNotFoundException e) {
95              e.printStackTrace();
96              throw e;
97          }
98      }
99
100     @Override
101     public List<MobileContact> getAllContacts() { return dao.getAll(); }
104
```

Delete contact με phone number


```
105     @Override
106     public void deleteContact(String phoneNumber) throws ContactNotFoundException {
107         try {
108             if (!dao.phoneNumberExists(phoneNumber)) {
109                 throw new ContactNotFoundException(phoneNumber);
110             }
111
112             dao.delete(phoneNumber);
113         } catch (ContactNotFoundException e) {
114             e.printStackTrace();
115             throw e;
116         }
117     }
```

Get contact με phone number


```
119     @Override
120     public MobileContact getContact(String phoneNumber) throws ContactNotFoundException {
121         MobileContact mobileContact;
122         try {
123             mobileContact = dao.get(phoneNumber);
124             if (mobileContact == null) {
125                 throw new ContactNotFoundException(phoneNumber);
126             }
127             return mobileContact;
128         } catch (ContactNotFoundException e) {
129             e.printStackTrace();
130             throw e;
131         }
132     }
```

Mappers (1)


```
134     /**
135      * Maps {@link MobileContactInsertDTO} to {@link MobileContact}.
136      *
137      * @param mobileContact
138      *          the {@link MobileContact} under creation.
139      * @param mobileContactInsertDTO
140      *          the Mobile Contact transfer object.
141      */
142 @   private void mapInsertMobileContact(MobileContact mobileContact, MobileContactInsertDTO mobileContactInsertDTO) {
143         mobileContact.setId(mobileContactInsertDTO.getId());
144         mobileContact.setPhoneNumber(mobileContactInsertDTO.getPhoneNumber());
145         UserDetails userDetails = new UserDetails();
146         mapUserDetails(userDetails, mobileContactInsertDTO.getUserDetailsDTO());
147         mobileContact.setUserDetails(userDetails);
148     }
149
150 @   private void mapUpdateMobileContact(MobileContact mobileContact, MobileContactUpdateDTO mobileContactUpdateDTO) {
151         mobileContact.setId(mobileContactUpdateDTO.getId());
152         mobileContact.setPhoneNumber(mobileContactUpdateDTO.getPhoneNumber());
153         UserDetails userDetails = new UserDetails();
154         mapUserDetails(userDetails, mobileContactUpdateDTO.getUserDetailsUpdateDto() );
155         mobileContact.setUserDetails(userDetails);
156     }
```

Mappers (2)


```
158     /**
159      * Maps {@link UserDetailsInsertDTO} to {@link UserDetails} .
160      *
161      * @param userDetails
162      *          the {@link UserDetails} under creation.
163      * @param userDetailsInsertDTO
164      *          the UserDetails Transfer object.
165      */
166 @   private void mapUserDetails(UserDetails userDetails, UserDetailsInsertDTO userDetailsInsertDTO) {
167         userDetails.setId(userDetailsInsertDTO.getId());
168         userDetails.setFirstname(userDetailsInsertDTO.getFirstname());
169         userDetails.setLastname(userDetailsInsertDTO.getLastname());
170     }
171
172 @   private void mapUserDetails(UserDetails userDetails, UserDetailsUpdateDTO userDetailsUpdateDTO) {
173         userDetails.setId(userDetailsUpdateDTO.getId());
174         userDetails.setFirstname(userDetailsUpdateDTO.getFirstname());
175         userDetails.setLastname(userDetailsUpdateDTO.getLastname());
176     }
177 }
```

Client

- Τις υπηρεσίες του Service Layer μπορεί να τις καλεί οποιοσδήποτε client (η main(), μία web σελίδα, ένα πρόγραμμα όπως JavaScript, ένα Android ή iOS app) από οποιαδήποτε συσκευή
- Τυπικά (όπως θα δούμε) σε enterprise εφαρμογές over Web μεταξύ του Client και του Service Layer παρεμβάλλεται ένας Web Server καθώς και ένα ακόμα Layer, ο Controller ο οποίος δρομολογεί τα requests προς το κατάλληλο service και αποστέλλει πίσω τα responses
- Σε αυτά τα setups ο Controller είναι η οντότητα που επικοινωνεί άμεσα με το Service Layer

Εργασία

- Αναπτύξτε μία εφαρμογή με Service Oriented Architecture (SOA) που να έχει ως Domain Model ένα Account με πεδία ίδια με το παράδειγμα Account που έχουμε δει.
- Αναπτύξτε όλα τα Layers καθώς και την main που θα λειτουργεί ως client