Αντικειμενοστρεφής Σχεδίαση και Domain model

Φροντιστήριο

Γιάννης Βασιλόπουλος, Ph.D.

Τμήμα Μηχανικών Η/Υ Και Πληροφορικής

Εισαγωγή

1. Επανάληψη στις έννοιες του OOP
2. Αντικειμενοστρεφής σχεδιασμός
3. UML – Class Diagram
4. Καλές πρακτικές
5. Αντιπαραδείγματα

Εισαγωγή

1. **Επανάληψη στις έννοιες του OOP**
2. Αντικειμενοστρεφής σχεδιασμός
3. UML – Class Diagram
4. Καλές πρακτικές
5. Αντιπαραδείγματα

Επανάληψη στις έννοιες του OOP

- Αντικειμενοστρεφής Προγραμματισμός
  - Αλλαγή θεώρησης στην αντίληψη του λογισμικού
  - Η εστίασή του ξεπερνά τις συναρτήσεις και διαδικασίες
  - Επικεντρώνεται σε Αντικείμενα που συνδυάζουν ιδιότητες με συμπεριφορές
    - Αυτό φαίνεται να αντικατοπτρίζει τον τρόπο που σκεφτόμαστε για τον κόσμο γύρω μας
  - Δημιουργεί μια ψηφιακή αναπαράσταση που ταιριάζει περισσότερο με το νοητικό μας μοντέλο
    - Ποια είναι τα «πράγματα» στο πεδίο του προβλήματός μας;
    - Τι χαρακτηριστικά έχουν αυτά τα πράγματα;
    - Ποιες συμπεριφορές παρουσιάζουν;
    - Πώς αλληλεπιδρούν αυτά τα πράγματα μεταξύ τους;

Επανάληψη στις έννοιες του OOP
Κλάσεις και Αντικείμενα

- Κλάση (ή Τάξη)
  - Σχέδιο ή πρότυπο που ορίζει τη δομή και τη συμπεριφορά των αντικειμένων. Χρησιμεύει ως ένας τύπος δεδομένων που ορίζεται από τον χρήστη και ορίζει δεδομένα (χαρακτηριστικά ή ιδιότητες) και μεθόδους που λειτουργούν σε αυτά τα δεδομένα
- Αντικείμενο
  - Ένα αντικείμενο προκύπτει από την υλοποίηση αυτών που περιγράφει μια κλάση. Είναι μια φυσική οντότητα που υπάρχει στη μνήμη, και έχει μια κατάσταση και μια συμπεριφορά που καθορίζονται από το σύνολο τιμών που έχει για τα χαρακτηριστικά που ορίζονται στην κλάση

Επανάληψη στις έννοιες του OOP
Βασικές έννοιες στον OOP (1)

- Ενθυλάκωση (ή Κελυφοποίηση)
  - Μειώνει την πολυπλοκότητα αποκρύπτοντας λεπτομέρειες υλοποίησης
  - Προστατεύει την ακεραιότητα των δεδομένων ελέγχοντας την πρόσβαση
  - Επιτρέπει την αλλαγή της υλοποίησης χωρίς να επηρεάζεται ολόκληρος ο κώδικας

Επανάληψη στις έννοιες του OOP
Βασικές έννοιες στον OOP (2)

- Κληρονομικότητα
  - Συνδέει κλάσεις με σχέση «is-a»επιτρέποντας σε μια παράγωγη κλάση (υποκλάση) κληρονομεί χαρακτηριστικά και συμπεριφορές από μια βασική κλάση (υπερκλάση)
  - Προωθεί την επαναχρησιμοποίηση κώδικα
  - Μοντελοποιεί φυσικές ιεραρχίες
  - Υποστηρίζει πολυμορφική συμπεριφορά

Επανάληψη στις έννοιες του OOP
Βασικές έννοιες στον OOP (3)

- Πολυμορφισμός
  - Αντικείμενα διαφορετικών κλάσεων αντιμετωπίζονται ως αντικείμενα μιας κοινής υπερκλάσης.
  - Δυνατότητα διαφορετικής επεξεργασίας αντικειμένων ανάλογα με τον τύπο δεδομένων ή την κλάση τους.
  - Τύποι πολυμορφισμού:
    - Υπερφόρτωση (Overloading) μεθόδων (Πολυμορφισμός σε χρόνο μεταγλώττισης)
      - Ίδιο όνομα μεθόδου, διαφορετικές παράμετροι
      - Επιλύεται κατά τη μεταγλώττιση
    - Επικάλυψη (Overriding) μεθόδων (Πολυμορφισμός σε χρόνο εκτέλεσης)
      - Ίδια υπογραφή μεθόδου, διαφορετικές υλοποιήσεις σε υποκλάσεις
      - Επιλύεται κατά την εκτέλεσης με βάση τον πραγματικό τύπο του αντικειμένου

Επανάληψη στις έννοιες του OOP
Βασικές έννοιες στον OOP (4)

- Αφαίρεση
  - Απόκρυψη πολύπλοκων λεπτομερειών υλοποίησης και παρουσίαση μόνο των βασικών χαρακτηριστικών ενός αντικειμένου
  - Απλουστευμένο μοντέλο που αποτυπώνει βασικές ιδιότητες και συμπεριφορές.
    - Μειώνει την πολυπλοκότητα
    - Επικεντρώνεται στο τι κάνει ένα αντικείμενο, όχι στο πώς το κάνει
    - Επιτρέπει αλλαγές στην υλοποίηση χωρίς να επηρεάζει τη διεπαφή
  - Επιτυγχάνεται με
    - Abstract class
    - Interface

Επανάληψη στις έννοιες του OOP
Βασικές έννοιες στον OOP (5)

- Abstract class
  - Υλοποιούνται μόνο από υποκλάσεις
  - Μπορούν να περιέχουν αφηρημένες και υλοποιημένες μεθόδους
  - Οι υποκλάσεις πρέπει να υλοποιούν όλες τις αφηρημένες μεθόδους
  - Χρήση:
    - Ίδιος κώδικας για πολλές στενά συσχετιζόμενες κλάσεις
    - Όταν πρέπει να δηλωθούν μη δημόσια μέλη
- Interface
  - Δέσμευση για υλοποίηση συγκεκριμένων μεθόδων και σταθερών (public, static, final)
  - Καμία λεπτομέρεια υλοποίησης
  - Οι κλάσεις μπορούν να υλοποιήσουν πολλαπλά interface
  - Χρήση:
    - Καθορισμός συμπεριφοράς για μια κλάση ανεξάρτητα από τη θέση της στην κληρονομική ιεραρχία
    - Χρειάζεται πολλαπλή κληρονομικότητα

1. Επανάληψη στις έννοιες του OOP
2. **Αντικειμενοστρεφής σχεδιασμός**
3. UML – Class Diagram
4. Καλές πρακτικές
5. Αντιπαραδείγματα

Αντικειμενοστρεφής σχεδιασμός

- Πώς σχεδιάζουμε τις κλάσεις μας;
- Σχεδιάζουμε τις κλάσεις μας ώστε να αναπτύσσουμε ευκολά συντηρήσιμο, επεκτάσιμο και εύρωστο λογισμικό.
- Για να το πέτυχουμε αυτό μπορούμε να ακολουθήσουμε κάποιες καλές πρακτικές και βασικές αρχές στην αντικειμενοστρεφή σχεδίαση

Αντικειμενοστρεφής σχεδιασμός
Καλές πρακτικές

- Μέγεθος και πολυπλοκότητα κλάσης
  - Το περιεχόμενο είναι πρωταρχικής σημασίας
  - Μερικές γενικές οδηγίες για την εκτίμηση του μεγέθους:
    - Οι κλάσεις καλό είναι γενικά να περιέχουν κάτω από 500 γραμμές κώδικα
    - Οι κλάσεις καλό είναι να μην έχουν περισσότερες από 20 μεθόδους
    - Οι μέθοδοι καλό είναι να είναι κάτω από 30 γραμμές
    - Αν δεν μπορείτε να περιγράψετε τι κάνει μια κλάση σε μια πρόταση, είναι μάλλον πολύ μεγάλη

Αντικειμενοστρεφής σχεδιασμός
Καλές πρακτικές

- Συνοχή (cohesion)
  - Η συνοχή σχετίζεται με τον αριθμό και την ποικιλία των αρμοδιοτήτων για τις οποίες είναι υπεύθυνη κάθε μονάδα (κλάσεις, μέθοδοι, και πακέτα) μιας εφαρμογής
  - Αν κάθε μονάδα κώδικα είναι υπεύθυνη για μία μόνο λογική εργασία, τότε λέμε ότι έχει υψηλή συνοχή
  - Γενικά επιδιώκουμε υψηλή συνοχή
  - Οφέλη:
    - Ο σκοπός της κλάσης είναι πιο ευκολά κατανοήτος.
    - Βελτιώνεται η δυνατότητα ελέγχου και επαναχρησιμοποίησης του κώδικα
    - Μειώνεται ο αντίκτυπος αλλαγών (οι αλλαγές περιορίζονται σε τοπικό επίπεδο)
    - Απλοποιείται η συντήρηση και αποσφαλμάτωση

Αντικειμενοστρεφής σχεδιασμός

Καλές πρακτικές

- Σύζευξη (coupling)
  - Η σύζευξη αναφέρεται στην εσωτερική σύνδεση μεταξύ ξεχωριστών μονάδων του προγράμματος.
  - Αν δύο κλάσεις εξαρτώνται στενά για πολλές λεπτομέρειες η μία από την άλλη, τότε λέμε ότι αυτές οι κλάσεις έχουν στενή σύζευξη
  - Γενικά επιδιώκουμε χαλαρή σύζευξη
  - Οφέλη:
    - Ο σκοπός της κλάσης είναι πιο ευκολά κατανοήσιμος χωρίς να είναι απαραίτητο να διαβάζουμε άλλες κλάσεις
    - Βελτιώνεται η δυνατότητα τροποποίησης μια κλάσης με μικρή ή και καθόλου επίδραση σε άλλες κλάσεις.
    - Γίνεται πιο ξεκάθαρη η ροή ελέγχου ανάμεσα σε αντικείμενα διαφορετικών κλάσεων
    - Αυξάνεται η συντηρησιμότητα

Αντικειμενοστρεφής σχεδιασμός
Καλές πρακτικές

- Μείωση της σύζευξης
  - Ενθυλάκωση
    - Δεν επιτρέπονται αναφορές σε μέλη τύπου private από το εξωτερικό της κλάσης
    - Μειώνεται η επίπτωση των εσωτερικών αλλαγών.
  - Καθοδηγούμενη από αρμοδιότητες σχεδίαση (responsibility-driven design)
    - Κάθε κλάση πρέπει να αναλαμβάνει να χειρίζεται τα δεδομένα της
    - Η κλάση στην οποία ανήκουν τα δεδομένα πρέπει να αναλαμβάνει και την επεξεργασία τους
    - Όταν πρέπει να γίνει κάποια αλλαγή, τότε πρέπει να επηρεάζονται όσο το δυνατόν λιγότερες κλάσεις
  - Χρήση interface
  - Επικοινωνία μόνο με τις άμεσα σχετιζόμενες κλάσεις

Αντικειμενοστρεφής σχεδιασμός
Βασικές αρχές σχεδιασμού κλάσεων

- Single Responsibility Principle (SRP)
  - Κάθε κλάση θα πρέπει να υπεύθυνη για μια μόνο αρμοδιότητα, δηλαδή να έχει μόνο μια αιτία για να αλλάξει
  - Κλάσεις με πολλές αρμοδιότητες
    - Κατανοούνται, συντηρούνται και ελέγχονται πιο δύσκολα
    - Αλλαγές σε μια αρμοδιότητα μπορεί να επηρεάσουν άλλες αρμοδιότητες
    - Μειώνεται η δυνατότητα επαναχρησιμοποίησης του κώδικά τους
  - Παράδειγμα:
    - Μια κλάση UserManager που αποθηκεύει δεδομένα χρηστών και χειρίζεται την αυθεντικοποίηση, έχει τουλάχιστον δύο αρμοδιότητες (λόγους να αλλάξει):
      - αλλαγές στον τρόπο που αποθηκεύονται τα δεδομένα των χρηστών
      - αλλαγές στη λογική αυθεντικοποίησης
    - Η εφαρμογή της αρχής SRP θα διαχώριζε τις αρμοδιότητες σε δύο κλάσεις

Αντικειμενοστρεφής σχεδιασμός
Βασικές αρχές σχεδιασμού κλάσεων

- Open-Closed Principle (OCP)
  - Οι κλάσεις πρέπει να είναι ανοικτές σε επεκτάσεις, αλλά όχι σε τροποποιήσεις, δηλαδή να μπορούμε να προσθέσουμε νέα λειτουργικότητα χωρίς να πειράξουμε τον κώδικά τους
  - Οφέλη:
    - Μειώνεται ο κίνδυνος εισαγωγής σφαλμάτων σε δοκιμασμένο κώδικα
    - Ασφαλέστερες αναβαθμίσεις και συντήρηση
    - Υποστηρίζεται η δομικότητα (modurality) και η ευκολότερη υλοποίηση νέων λειτουργιών
  - Σχεδιασμός των τμημάτων που είναι πιθανό να αλλάξουν με δυνατότητα επέκτασης

Αντικειμενοστρεφής σχεδιασμός
Βασικές αρχές σχεδιασμού κλάσεων

- Παράδειγμα μη εφαρμογής OCP

```java
// Poor implementation that violates OCP
public class PaymentProcessor {
    public void processPayment(String type, double amount) {
        if (type.equals("CREDIT_CARD")) {
            // Credit card processing logic
        } else if (type.equals("PAYPAL")) {
            // PayPal processing logic
        } else if (type.equals("BANK_TRANSFER")) {
            // Bank transfer logic
        }
        // Adding a new payment type requires modifying this class
    }
}
```

Αντικειμενοστρεφής σχεδιασμός

Βασικές αρχές σχεδιασμού κλάσεων

- Παράδειγμα εφαρμογής OCP

```java
// Abstract payment processor
public abstract class PaymentProcessor {
    public abstract void processPayment(double amount);
}

// Concrete implementations
public class CreditCardProcessor extends PaymentProcessor {
    @Override
    public void processPayment(double amount) {
        // Credit card processing logic
    }
}
```

```java
public class PayPalProcessor extends PaymentProcessor {
    @Override
    public void processPayment(double amount) {
        // PayPal processing logic
    }
}

// Adding a new payment type just requires creating a new class
public class CryptoCurrencyProcessor extends PaymentProcessor {
    @Override
    public void processPayment(double amount) {
        // Cryptocurrency processing logic
    }
}
```

Αντικειμενοστρεφής σχεδιασμός
Βασικές αρχές σχεδιασμού κλάσεων

- Liskov Substitution Principle (LSP)
  - Τα αντικείμενα μιας υπερκλάσης θα πρέπει να μπορούν να αντικατασταθούν με αντικείμενα των υποκλάσεών της χωρίς να επηρεάζεται η ορθότητα του προγράμματος
  - Οφέλη:
    - Καλά σχεδιασμένες ιεραρχίες κληρονομικότητας
    - Αποτρέπεται μη αναμενόμενη συμπεριφορά κατά τη χρήση πολυμορφισμού
    - Υλοποιείται ουσιαστικά η σχέση «is-a»
    - Αξιόπιστη επαναχρησιμοποίηση κώδικα
  - Μπορεί να εφαρμοστεί και σε interface και abstract class

Αντικειμενοστρεφής σχεδιασμός

Βασικές αρχές σχεδιασμού κλάσεων

- Παράδειγμα LSP

```
public abstract class BankAccount
{
   public boolean withDraw(double amount);

   public void deposit(double amount);

}
```

Όλα ΟΚ! Η υποκλάση ορίζεται σωστά

```
public class BasicAccount extends BankAccount
{
    private double balance;

    @Override
    public boolean withDraw(double amount)
    {
      balance -= amount
    }

    @Override
    public void deposit(double amount)
    {
       balance += amount;
    }
}
```

Αντικειμενοστρεφής σχεδιασμός

Βασικές αρχές σχεδιασμού κλάσεων

- Παράδειγμα LSP

```java
public abstract class BankAccount
{
   public boolean withDraw(double amount);

   public void deposit(double amount);

}
```

Η υποκλάση έχει συμπεριφορά που δεν έχει οριστεί στην υπερκλάση

```java
// Withdraws are not permitted!
public class InvestmentAccount extends BankAccount
{
    private double balance;

    @Override
    public boolean withDraw(double amount)
    {
      throw new Expcetion("Not supported");
    }

    @Override
    public void deposit(double amount)
    {
       balance += amount;
    }
}
```

Αντικειμενοστρεφής σχεδιασμός
Βασικές αρχές σχεδιασμού κλάσεων

- Interface Segregation Principle (ISP)
  - Κλάσεις δεν θα πρέπει να εξαρτώνται από interface που δεν χρησιμοποιούν
    - Πολλά και μικρότερα αντί για λίγα και μεγάλα γενικού σκοπού interface
  - Οφέλη
    - Μειώνεται ο αντίκτυπος των αλλαγών (αλλαγή σε ένα interface επηρεάζει λιγότερες κλάσεις)
    - Οι κλάσεις δεν υλοποιούν μεθόδους που δεν χρειάζονται
    - Υποστηρίζεται η χαλαρή συζευξή μεταξύ των αντικειμένων
  - Τυφλή εφαρμογή: πάρα πολλά και πολύ μικρά interface
  - Ισορροπημένη προσέγγιση: ομαδοποίηση μεθόδων που αλλάζουν από κοινού

Αντικειμενοστρεφής σχεδιασμός
Βασικές αρχές σχεδιασμού κλάσεων

- Παράδειγμα μη εφαρμογής ISP

```
// Problematic large interface
public interface MultiFunctionDevice {
    void print(Document d);
    void scan(Document d);
    void fax(Document d);
    void copy(Document d);
    void staple(Document d);
    void bind(Document d);
}
```

Αντικειμενοστρεφής σχεδιασμός

Βασικές αρχές σχεδιασμού κλάσεων

- Παράδειγμα εφαρμογής ISP

```java
// Segregated interfaces
public interface Printer {
    void print(Document d);
}

public interface Scanner {
    void scan(Document d);
}

public interface FaxMachine {
    void fax(Document d);
}

public interface Copier {
    void copy(Document d);
}
```

```java
// Classes implement only what they need
public class SimplePrinter implements Printer {
    public void print(Document d) { ... }
}

public class HomeOfficePrinter implements Printer, Scanner, Copier {
    public void print(Document d) { ... }
    public void scan(Document d) { ... }
    public void copy(Document d) { ... }
}

public class EnterpriseDevice implements Printer, Scanner, Copier, FaxMachine {
    // Implements all methods
}
```

Αντικειμενοστρεφής σχεδιασμός
Βασικές αρχές σχεδιασμού κλάσεων

- Dependency Inversion Principle (DIP)
  - Οι κλάσεις υψηλού επιπέδου (εφαρμόζουν business logic) δεν πρέπει να εξαρτώνται από κλάσεις χαμηλού επιπέδου (υλοποιούν βασικές λειτουργίες), αλλά και οι δύο πρέπει να εξαρτώνται από interface
  - Οφέλη
    - Αλλαγές στις υλοποιήσεις χωρίς να επηρεάζονται οι κλάσεις υψηλού επιπέδου.
    - Ο κώδικας γίνεται πιο κατανοητός και εύκολος στην τροποποίηση
    - Τρόπος για την εφαρμογή της Open-Closed Principle
  - Εφαρμογή και σε υψηλότερο επίπεδο αφαίρεσης, όπως module, πακέτα, κ.τ.λ
  - Τεχνική Dependency Injection: πέρασμα εξαρτήσεων σε constructors, setters, κ.τ.λ.

Αντικειμενοστρεφής σχεδιασμός
Βασικές αρχές σχεδιασμού κλάσεων

- Παράδειγμα μη εφαρμογής DIP

```
public class ReportGenerator {
    private Database database;

    public ReportGenerator() {
        this.database = new Database(); // Dependency on Database Class
    }

    public void generateReport() {
        // Use database for data retrieval
    }
}

public class Database {
    // Code for connecting to and accessing database
}
```

Αντικειμενοστρεφής σχεδιασμός
Βασικές αρχές σχεδιασμού κλάσεων

- Παράδειγμα εφαρμογής DIP

```
public class ReportGenerator {
    private DataSource dataSource; // Dependency on DataSource Interface

    public ReportGenerator(DataSource dataSource) {
        this.dataSource = dataSource;
    }

    public void generateReport() {
        // Use dataSource for data retrieval
    }
}

public interface DataSource {
    // Methods for data retrieval
}

public class Database implements DataSource {
    // Code for connecting to and accessing database
}
```

Αντικειμενοστρεφής σχεδιασμός

Συνέργεια αρχών για τον σχεδιασμό κλάσεων

- Σχεδιασμός SOLID
- SRP: Η δημιουργία κλάσεων με μια μοναδική αρμοδιότητα, διευκολύνει την επέκταση (OCP), την αντικατάσταση (LSP) και την ενσωμάτωση μέσω interface (ISP)
- OCP: Ο σχεδιασμός κλάσεων που είναι ανοιχτές σε επέκταση συχνά περιλαμβάνει τη δημιουργία interface, δηλαδή εφαρμογή DIP
- LSP: Ο σχεδιασμός για σωστή αντικατάσταση υπερκλασεων-υποκλεσεων είναι απαραίτητός για την αποτελεσματική λειτουργία της OCP
- ISP: Η δημιουργία interface υποστηρίζει τόσο την SRP (σε επίπεδο interface) όσο και την DIP
- DIP: Η εξάρτηση από interface είναι βασικός μηχανισμός για την εφαρμογή της OCP

Αντικειμενοστρεφής σχεδιασμός

Κόστος εφαρμογής των αρχών για τον σχεδιασμό κλάσεων

- Η εφαρμογή αυτών των αρχών έχει κόστος: περισσότερες κλάσεις, περισσότερα interface και μεγαλύτερη πολυπλοκότητα στην αρχική σχεδίαση
- Γι' αυτό:
  - Εφαρμόστε τις αρχές όπου αναμένονται αλλαγές ή όπου απαιτείται ευελιξία
  - Εξετάστε την κλίμακα και τη διάρκεια ζωής του έργου σας
  - Εξισορροπήστε τις αρχές με τους πρακτικούς περιορισμούς
  - Ξεκινήστε απλά και εφαρμόστε τις αρχές ανάλογα με τις ανάγκες
- Αυτές οι αρχές λειτουργούν περισσότερο σαν **κατευθυντήριες οδηγίες**, παρά σαν αυστηρούς κανόνες
- Στόχος: η δημιουργία λογισμικού που να είναι συντηρήσιμο, επεκτάσιμο και εύρωστο

1. Επανάληψη στις έννοιες του OOP
2. Αντικειμενοστρεφής σχεδιασμός
3. **UML – Class Diagram**
4. Καλές πρακτικές
5. Αντιπαραδείγματα

UML

Class Diagram

- UML (ενοποιημένη γλώσσα μοντελοποίησης)
  - Τυποποιημένη γλώσσα μοντελοποίησης που βοηθά τους προγραμματιστές να απεικονίζουν, να προσδιορίζουν, να κατασκευάζουν και να τεκμηριώνουν συστήματα λογισμικού
- Class Diagram (διάγραμμα κλάσεων)
  - Τύπος διαγραμμάτων της UML για τον αντικειμενοστρεφή σχεδιασμό
  - Χρήση:
    - Οπτικοποίηση της δομής των κλάσεων και των μεταξύ τους σχέσεων
    - Σχεδιασμός της αρχιτεκτονικής του συστήματός πριν την συγγραφή κώδικά
    - Τεκμηρίωση υφιστάμενων συστημάτων
    - Παρουσίαση σχεδιασμού μεταξύ των μελών της ομάδας
  - Εμείς ενημερώνουμε σε κάθε φάση το Domain Model για να καταλήξουμε στο Class Diagram

UML

Class Diagram

- UML (ενοποιημένη γλώσσα μοντελοποίησης)
  - Τυποποιημένη γλώσσα μοντελοποίησης που βοηθά τους προγραμματιστές να απεικονίζουν, να προσδιορίζουν, να κατασκευάζουν και να τεκμηριώνουν συστήματα λογισμικού
- Class Diagram (διάγραμμα κλάσεων)
  - Τύπος διαγραμμάτων της UML για τον αντικειμενοστρεφή σχεδιασμό
  - Χρήση:
    - Οπτικοποίηση της **δομής των κλάσεων** και των μεταξύ τους σχέσεων
    - Σχεδιασμός της αρχιτεκτονικής του συστήματός πριν την συγγραφή κώδικά
    - Τεκμηρίωση υφιστάμενων συστημάτων
    - Παρουσίαση σχεδιασμού μεταξύ των μελών της ομάδας
  - Εμείς ενημερώνουμε σε κάθε φάση το Domain Model για να καταλήξουμε στο Class Diagram

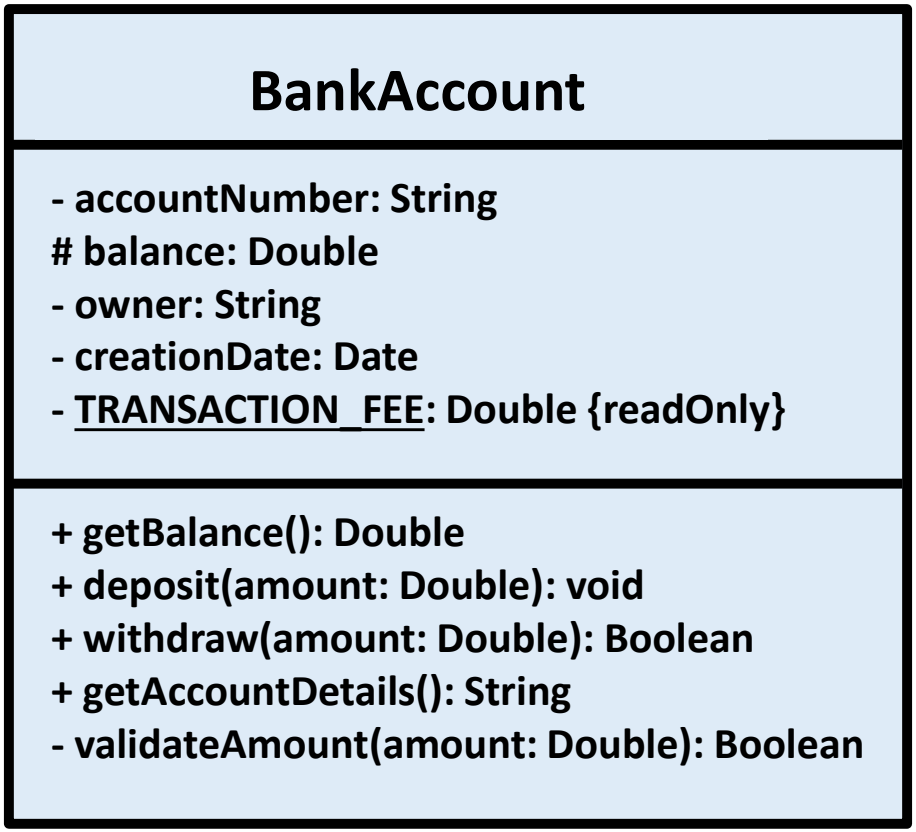
UML

Σημειογραφία Class Diagram

- Ορατότητα
  - - private
  - + public
  - # protected
  - ~ package
- staticAttribute
- Final: ΚΕΦΑΛΑΙΑ ή {readOnly}
- *Abstract κλάση ή μέθοδος*
- methodName(param: Type): ReturnType

UML

Παράδειγμα απεικόνισης κλάσης σε Class Diagram



UML

Class Diagram

- UML (ενοποιημένη γλώσσα μοντελοποίησης)
  - Τυποποιημένη γλώσσα μοντελοποίησης που βοηθά τους προγραμματιστές να απεικονίζουν, να προσδιορίζουν, να κατασκευάζουν και να τεκμηριώνουν συστήματα λογισμικού
- Class Diagram (διάγραμμα κλάσεων)
  - Τύπος διαγραμμάτων της UML για τον αντικειμενοστρεφή σχεδιασμό
  - Χρήση:
    - Οπτικοποίηση της **δομής των κλάσεων** και των μεταξύ τους σχέσεων
    - Σχεδιασμός της αρχιτεκτονικής του συστήματός πριν την συγγραφή κώδικά
    - Τεκμηρίωση υφιστάμενων συστημάτων
    - Παρουσίαση σχεδιασμού μεταξύ των μελών της ομάδας
  - Εμείς ενημερώνουμε σε κάθε φάση το Domain Model για να καταλήξουμε στο Class Diagram

UML

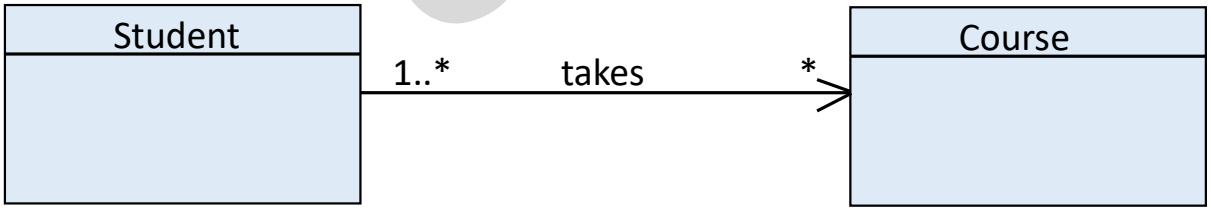
Class Diagram

- UML (ενοποιημένη γλώσσα μοντελοποίησης)
  - Τυποποιημένη γλώσσα μοντελοποίησης που βοηθά τους προγραμματιστές να απεικονίζουν, να προσδιορίζουν, να κατασκευάζουν και να τεκμηριώνουν συστήματα λογισμικού
- Class Diagram (διάγραμμα κλάσεων)
  - Τύπος διαγραμμάτων της UML για τον αντικειμενοστρεφή σχεδιασμό
  - Χρήση:
    - Οπτικοποίηση της δομής των κλάσεων και των **μεταξύ τους σχέσεων**
    - Σχεδιασμός της αρχιτεκτονικής του συστήματός πριν την συγγραφή κώδικά
    - Τεκμηρίωση υφιστάμενων συστημάτων
    - Παρουσίαση σχεδιασμού μεταξύ των μελών της ομάδας
  - Εμείς ενημερώνουμε σε κάθε φάση το Domain Model για να καταλήξουμε στο Class Diagram

UML

Σχέσεις μεταξύ των κλάσεων σε Class Diagram

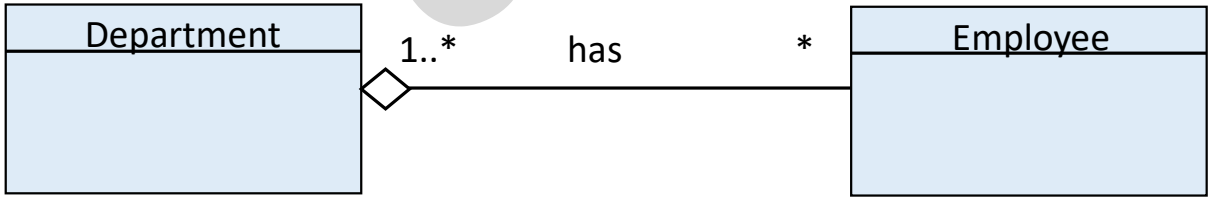
- Association (Συσχέτιση)
  - Η συσχέτιση αντιπροσωπεύει μια γενική σχέση μεταξύ κλάσεων
    - Δείχνει ότι αντικείμενα μιας κλάσης συνδέονται με αντικείμενα μιας άλλης κλάσης.
- Χαρακτηριστικά:
    - αντιπροσωπεύει μια σχέση «uses» ή «has-a».
    - μπορεί να είναι αμφίδρομη ή μονόδρομη
    - μπορεί να περιλαμβάνει πληθικότητα (το πλήθος των αντικείμενων που εμπλέκονται)
    - τα αντικείμενα έχουν το δικό τους κύκλο ζωής
  - Παράδειγμα:



UML

Σχέσεις μεταξύ των κλάσεων σε Class Diagram

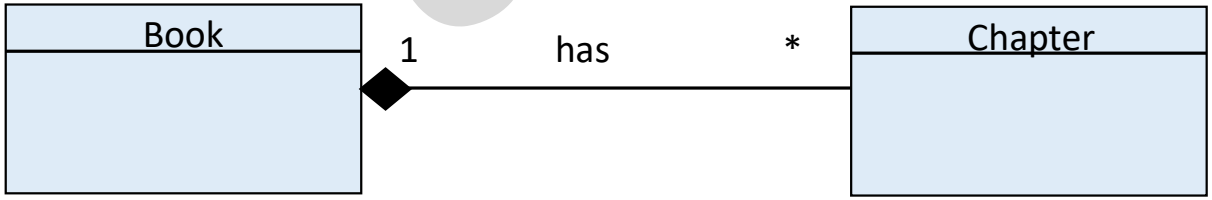
- Aggregation (Συνάθροιση)
  - Η συνάθροιση είναι μια εξειδικευμένη μορφή συσχέτισης που αντιπροσωπεύει μια σχέση «όλου/μέρους»
  - Χαρακτηριστικά:
    - αντιπροσωπεύει μια σχέση «has-a».
    - το «μέρος» μπορεί να υπάρχει ανεξάρτητα από το «όλο»
    - ένα «μέρος» μπορεί να συνδέεται με ένα «όλο» ή και με περισσότερα
    - αν το «όλο» πάψει να υπάρχει, τα «μέρη» συνεχίζουν να υπάρχουν ανεξάρτητα
  - Παράδειγμα:



UML

Σχέσεις μεταξύ των κλάσεων σε Class Diagram

- Composition (Σύνθεση)
  - Η σύνθεση είναι μια πιο ισχυρή μορφή συνάθροισης στην οποία το «μέρος» δεν μπορεί να υπάρξει χωρίς το «όλο»
  - Χαρακτηριστικά:
    - αντιπροσωπεύει μια ισχυρή σχέση «has-a».
    - το «μέρος» δεν μπορεί να υπάρχει ανεξάρτητα από το «όλο»
    - ένα «μέρος» συνδέεται μόνο με ένα «όλο»
    - αν το «όλο» πάψει να υπάρχει, τα «μέρη» παύουν και αυτά να υπάρχουν
  - Παράδειγμα:

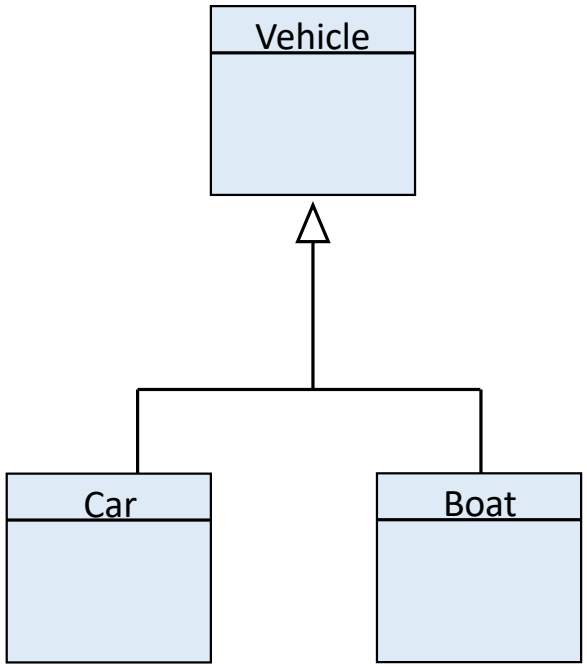


UML

Σχέσεις μεταξύ των κλάσεων σε Class Diagram

- Inheritance (Κληρονομικότητα - Γενίκευση)
  - Η κληρονομικότητα αντιπροσοπευει μια σχέση «is-a» ανάμεσα σε κλάσεις
  - Χαρακτηριστικά:
    - Η υποκλάση
      - κληρονομεί ιδιότητες και μεθόδους από την υπερκλάση
      - μπορεί να προσθέσει νέες ιδιότητες και μεθόδους
      - μπορεί να επικαλύψει τις μεθόδους (και τις ιδιότητες) της υπερκλάσης
    - Αντιπροσωπεύει την εξειδίκευση/γενίκευση

Παράδειγμα:

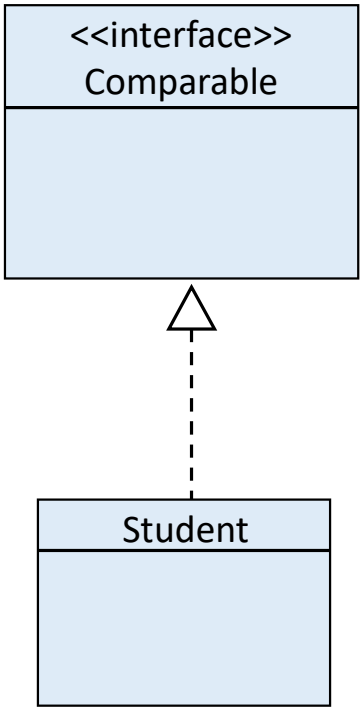


UML

Σχέσεις μεταξύ των κλάσεων σε Class Diagram

- Implementation (Υλοποίηση)
  - Η υλοποίηση αντιπροσωπεύει μια κλάση που υλοποιεί ένα interface
  - Χαρακτηριστικά:
    - Η κλάση πρέπει να υλοποιεί όλες τις μεθόδους που ορίζονται στο interface
    - Αντιπροσωπεύει μια δέσμευση για υλοποίηση συγκεκριμένων μεθόδων και σταθερών που εκπληρώνει η κλάση
    - Επιτρέπει τον πολυμορφισμό και τη χαλαρή σύζευξη

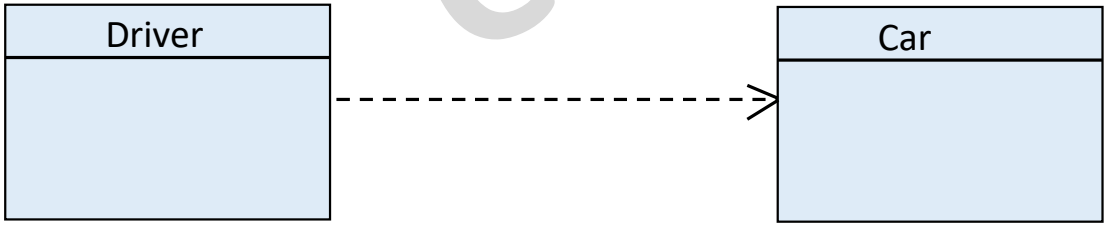
Παράδειγμα:



UML

Σχέσεις μεταξύ των κλάσεων σε Class Diagram

- Dependency (Εξάρτηση)
  - Εξάρτηση μεταξύ δύο κλάσεων υπάρχει όταν η μία κλάση βασίζεται στην άλλη, αλλά με πιο αδύναμη σχέση
    - Αντιπροσωπεύει μια χαλαρή σύζευξη (loose coupling) μεταξύ κλάσεων
  - Χαρακτηριστικά:
    - μια κλάση εξαρτάται από μια άλλη κλάση προσωρινά
    - μια κλάση χρησιμοποιεί μια άλλη κλάση ως παράμετρο εισόδου ή για να πραγματοποιήσει μια λειτουργία.
    - μια κλάση δημιουργεί ένα αντικείμενο μιας άλλης κλάσης μέσα σε μια μέθοδο.
    - η εξαρτημένη κλάση μπορεί να επηρεαστεί από αλλαγές στην κλάση που χρησιμοποίει
  - Παράδειγμα:



1. Επανάληψη στις έννοιες του OOP
2. Αντικειμενοστρεφής σχεδιασμός
3. UML – Class Diagram
4. **Καλές πρακτικές**
5. Αντιπαραδείγματα

UML

Καλές πρακτικές για την σχεδίαση Class Diagram

- Γενικές οδηγίες:
  - Μην ξεχνάτε τον σκοπό που σχεδιάζετε το class diagram
  - Φροντίστε το διάγραμμα να είναι καθαρό και ευανάγνωστό
  - Δεν χρειάζεται να δείχνει τα πάντα
  - Επικεντρωθείτε στις πιο σημαντικές σχέσεις και ιδιότητες
  - Χρησιμοποιήστε περιγραφικά και κατανοητά ονόματα στα Αγγλικά για κλάσεις, ιδιότητές και μεθόδους σύμφωνα με τις συμβάσεις ονοματολογίας
  - Χρησιμοποιήστε την τυποποιημένη σημειογραφία της UML
    - Διαφορετικά, αναφέρετε τον λόγο και την πηγή για χρήση άλλης σημειογραφίας
  - Κάντε χρήση των σωστών σχέσεων μεταξύ κλάσεων
    - Σε ειδικές περιπτώσεις και αν δεν είστε σίγουροι, χρησιμοποιήστε association
  - Μην ξεχνάτε την λεκτική περιγραφή των κλάσεων

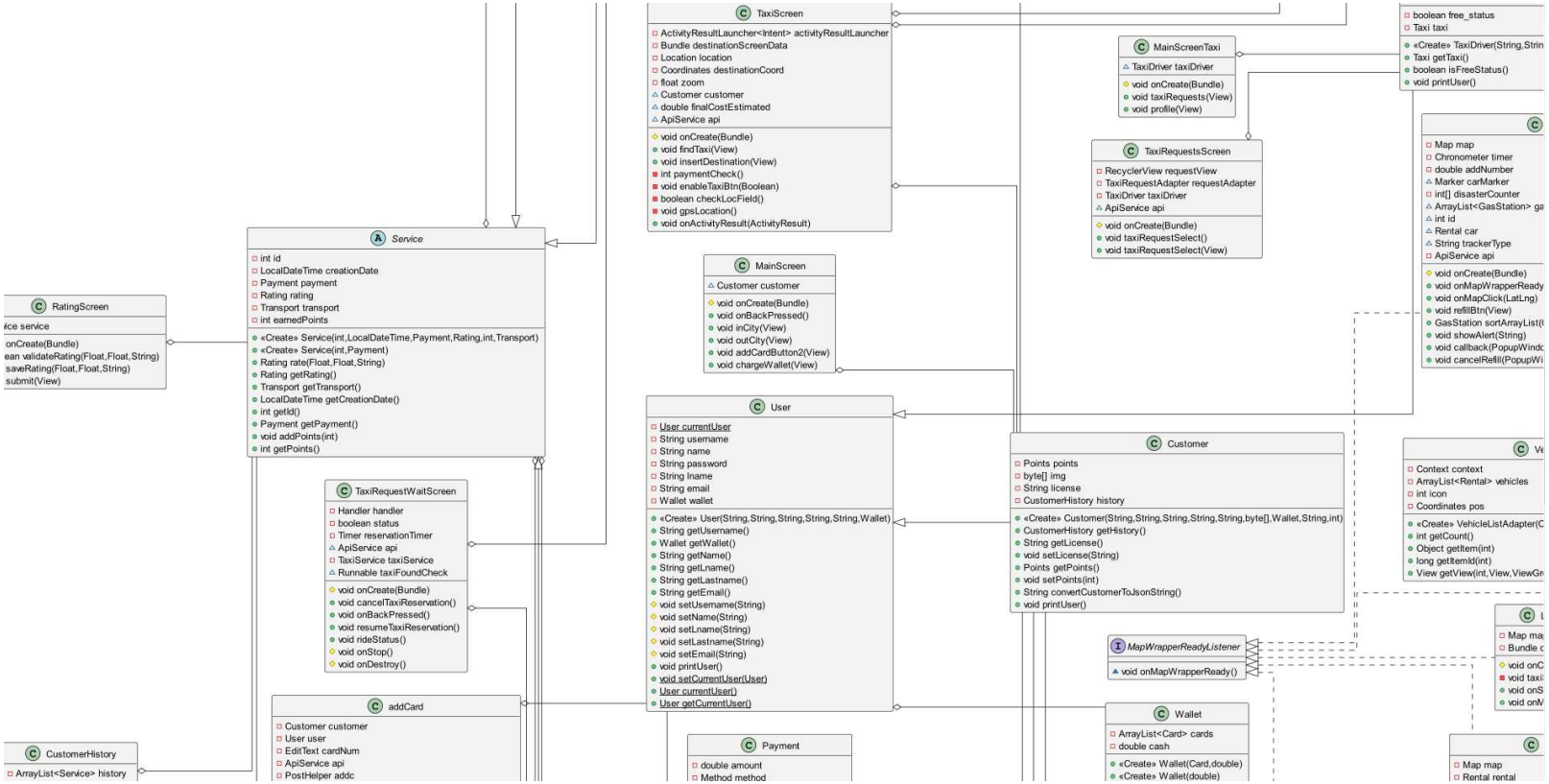
UML

Καλές πρακτικές για την σχεδίαση Class Diagram

- Χωροταξική οργάνωση:
  - Οργανώστε τις κλάσεις λογικά, συνήθως με τις κλάσεις υψηλότερου επιπέδου αφαίρεσης στην κορυφή
  - Ομαδοποιήστε συναφείς κλάσεις
  - Αφήστε χώρο για τον οπτικό διαχωρισμό των λογικών ομάδων
  - Χρησιμοποιήστε την τυποποιημένη σημειογραφία για την οπτική αναπαράσταση των διαφορετικών τύπων σχέσεων
  - Σχεδιάστε ξεκάθαρα και ευανάγνωστα τις σχέσεις μεταξύ των κλάσεων
  - Χρησιμοποιήστε ένα χρώμα για όλες τις κλάσεις του GUI

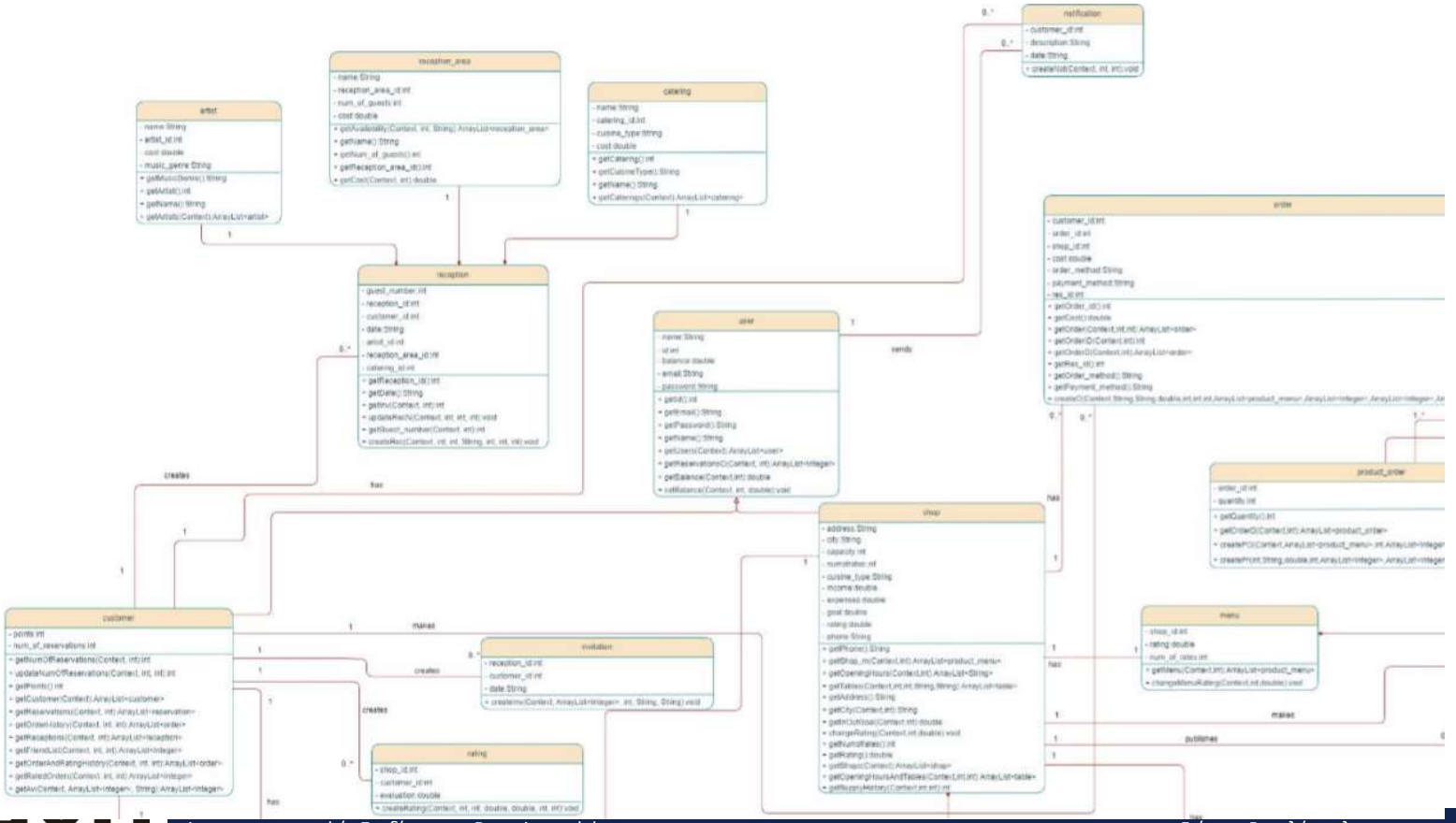
UML

Τμήμα από ένα σχετικά καλό Class Diagram (1)



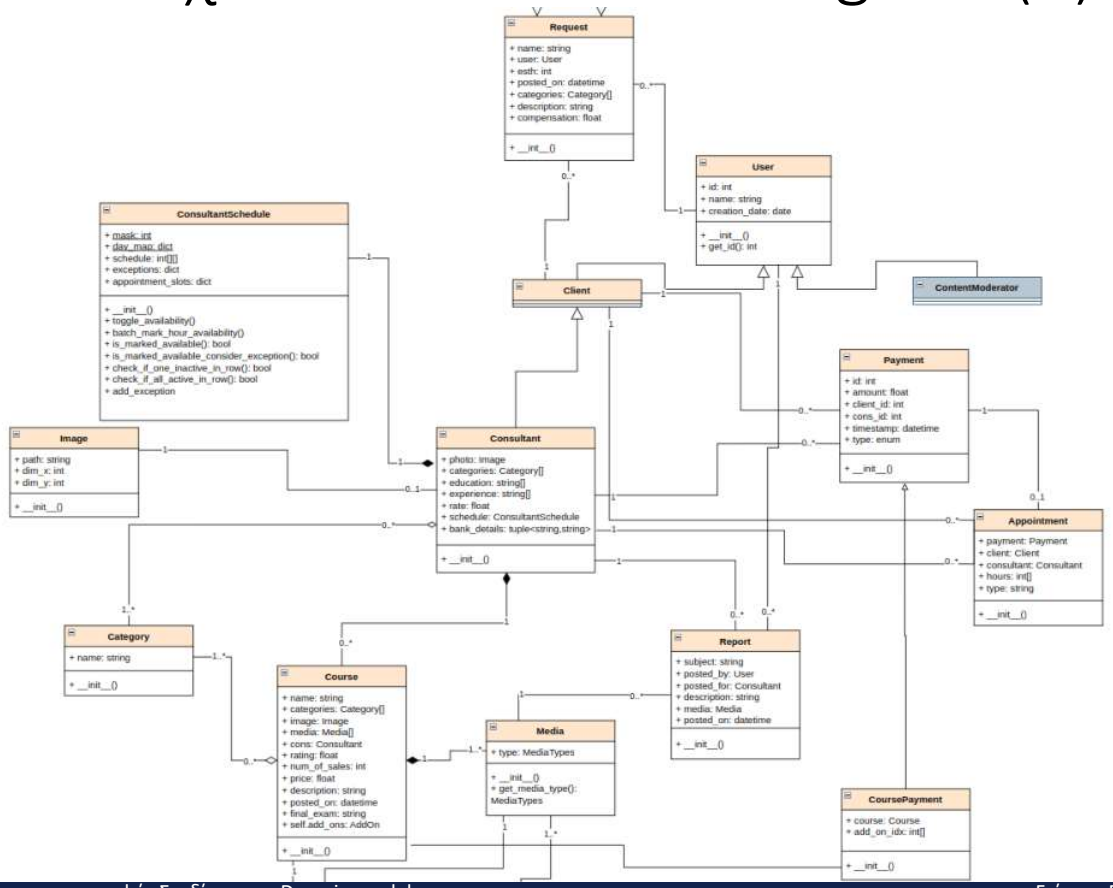
UML

Τμήμα από ένα σχετικά καλό Class Diagram (2)



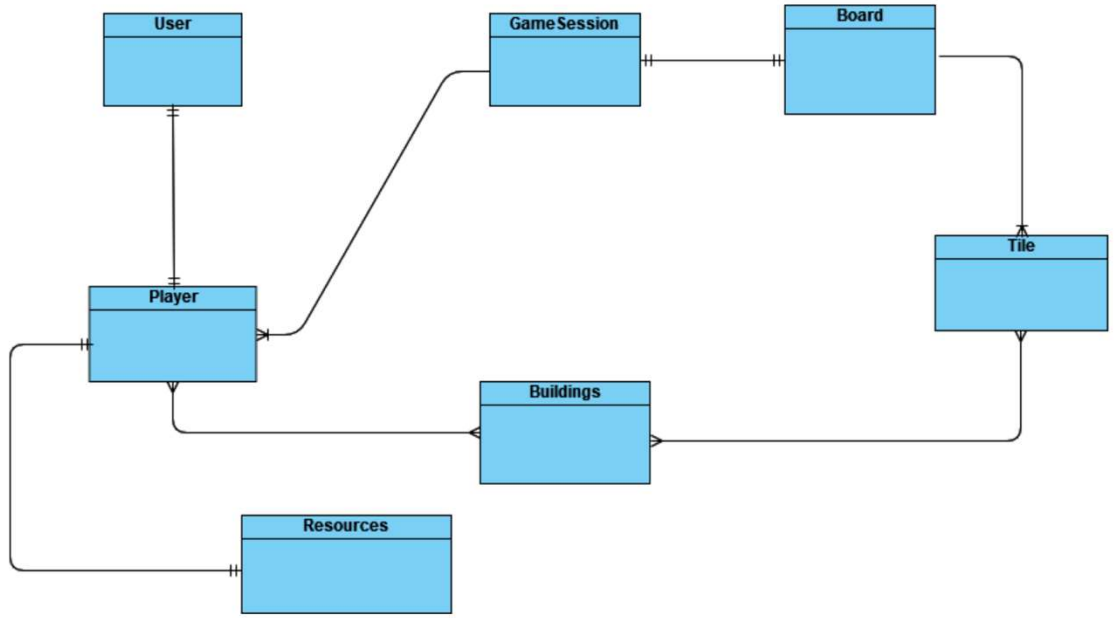
UML

Τμήμα από ένα σχετικά καλό Class Diagram (3)



1. Επανάληψη στις έννοιες του OOP
2. Αντικειμενοστρεφής σχεδιασμός
3. UML – Class Diagram
4. Καλές πρακτικές
5. **Αντιπαραδείγματα**

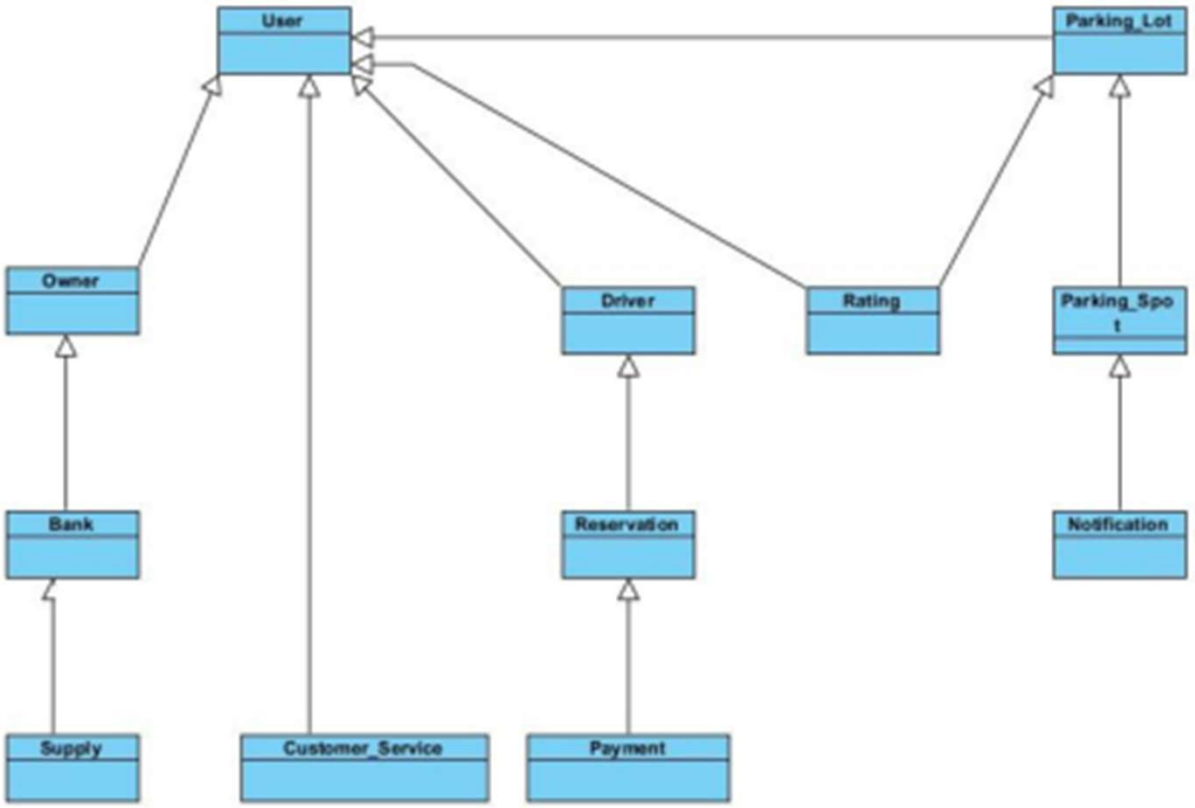
Λάθος σημειογραφία για domain model



Εικόνα 1 Διάγραμμα Domain Model

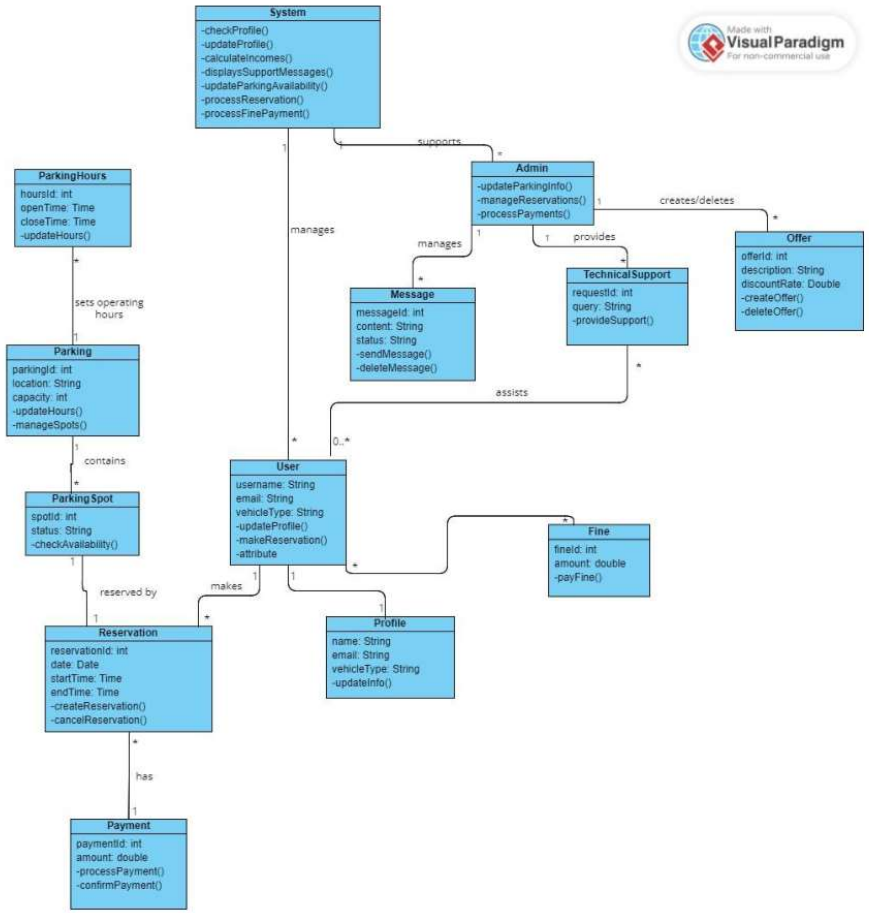
Οι κλάσεις συνδέονται μόνο με σχέσεις κληρονομικότητας

- Το διάγραμμα δεν παρουσιάζει λογικές συνδέσεις



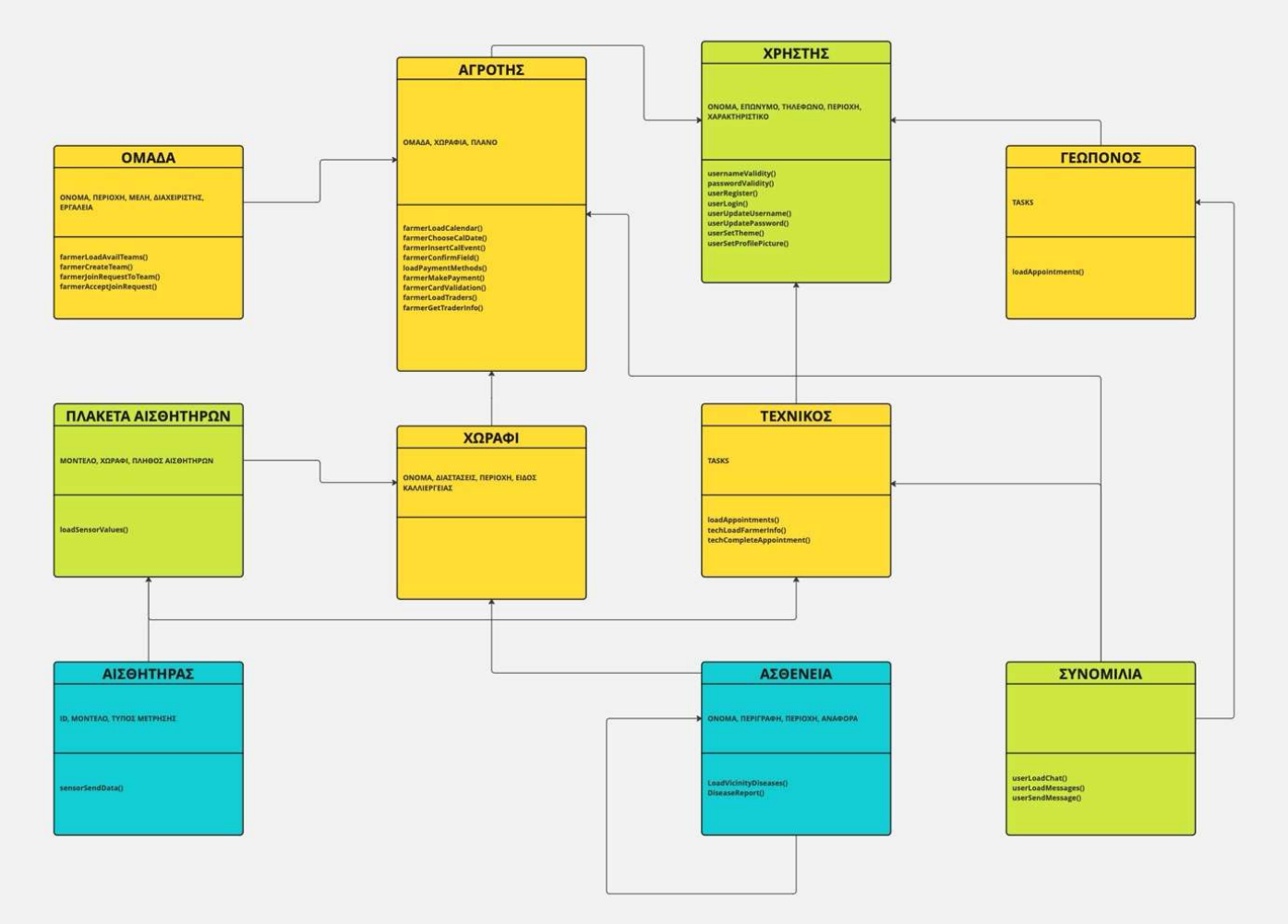
Χωρίς αντικειμενοστρεφή σχεδίαση

Διάγραμμα Domain-model:



Domain model με τεχνικά ζητήματα

- Ελληνικά και Αγγλικά
- Κλάσεις χωρίς μεθόδους ή ιδιότητες
- Λάθος σημειογραφία



Σύνοψη

- Η αντικειμενοστρεφής σχεδίαση είναι μια μεθοδολογία σχεδίασης λογισμικού που εστιάζει στην οργάνωση του λογισμικού γύρω από "αντικείμενα" που συνδυάζουν δεδομένα και συμπεριφορά
- Προσφέρει πολλά πλεονεκτήματα, όπως μειωμένη πολυπλοκότητα, επαναχρησιμοποίηση κώδικα, συντηρησιμότητα, επεκτασιμότητα και ανθεκτικότητα
- Οι αρχές SOLID είναι ένα σύνολο πέντε βασικών αρχών που βοηθούν στη δημιουργία καλών αντικειμενοστρεφών σχεδίων
- Τα διαγράμματα κλάσεων είναι ένα σημαντικό εργαλείο, καθώς βοηθούν στην οπτικοποίηση της δομής ενός συστήματος λογισμικού
- Η εφαρμογή καλών πρακτικών και τη δημιουργία διαγραμμάτων κλάσεων μπορεί να βοηθήσει στην ανάπτυξη ποιοτικότερού και πιο εύρωστου λογισμικού

Ευχαριστώ για την προσοχή σας

Απορίες;

Διευκρινήσεις;