Πρότυπα Σχεδίασης

Design Patterns

Πρότυπα Σχεδίασης

Όλα ξεκίνησαν από …..

την Αρχιτεκτονική !!

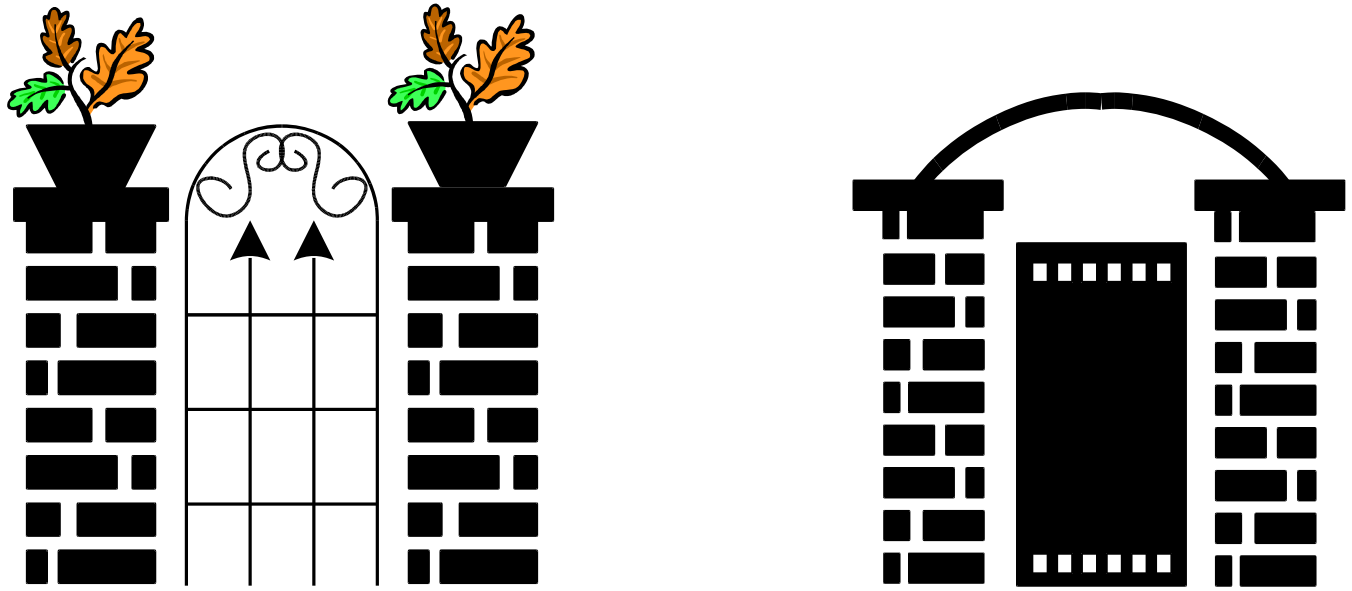
- Christopher Alexander, *The Timeless Way of Building*, Oxford University Press, New York, 1979: "Είναι η ποιότητα μία αντικειμενική ιδιότητα? "
- Αν αποδεχθεί κανείς ότι είναι δυνατό να αναγνωρίσει και να περιγράψει ένα σχέδιο καλής ποιότητας τότε:

(Alexander): Τι υπάρχει σε ένα σχέδιο καλής ποιότητας το οποίο δεν υπάρχει σε ένα σχέδιο κακής ποιότητας ?

- Οι καλές κατασκευές είχαν κοινά στοιχεία μεταξύ τους
- Τα κοινά στοιχεία συνήθως αφορούν κοινές λύσεις ή λύσεις σε κοινά προβλήματα
- Οι δομές δεν μπορούσαν να διαχωριστούν από το πρόβλημα το οποίο προσπαθούν να επιλύσουν.
  - Για το λόγο αυτό αναζήτησε διαφορετικές δομές που σχεδιάστηκαν για να επιλύσουν το ίδιο πρόβλημα

Πρότυπα Σχεδίασης

Λύσεις στο πρόβλημα της οριοθέτησης και ανάδειξης μιας εισόδου:



Ο Alexander ονόμασε τα κοινά στοιχεία μεταξύ των σχεδίων υψηλής ποιότητας πρότυπα (patterns)

Πρότυπα Σχεδίασης

Ο Alexander όρισε την έννοια του προτύπου ως "*Μία λύση ενός προβλήματος μέσα σε συγκεκριμένο πλαίσιο*" (…κάθε πρότυπο περιγράφει ένα πρόβλημα που εμφανίζεται συνέχεια στο περιβάλλον και στη συνέχεια περιγράφει τον πυρήνα της λύσης κατά τέτοιο τρόπο ώστε η λύση να μπορεί να εφαρμοστεί εκατομμύρια φορές)

…κατά ανάλογο τρόπο μία αυλή, ορθά σχηματισμένη, επιτρέπει στους ανθρώπους να τη ζήσουν.

… κυρίως, σε μία αυλή, οι άνθρωποι αναζητούν έναν ιδιωτικό εξωτερικό χώρο για να κάθονται κάτω από τον ουρανό, να χαίρονται τον ήλιο….

ένα Πρότυπο έχει πάντοτε ένα όνομα και ένα σκοπό

παρόλο που μπορεί να είναι μερικές φορές προφανές, είναι σημαντικό να διατυπώνεται ρητά το πρόβλημα που επιλύεται

Πρότυπα Σχεδίασης

...αν μία αυλή είναι στενή και δεν έχει θέα προς τα έξω, οι άνθρωποι αισθάνονται άβολα και απομακρύνονται ... χρειάζονται να βλέπουν προς κάτι μεγαλύτερο και μακρινό

... αν οι άνθρωποι περνούν από την αυλή καθημερινά ... η αυλή γίνεται οικεία και ένα φυσικό μέρος ... και χρησιμοποιείται

επισημαίνονται δυσκολίες με την απλή λύση και στη συνέχεια δίνονται οδηγίες για την επίλυση του προβλήματος που επισημάνθηκε

Η αξία ενός προτύπου είναι ότι με μικρή εμπειρία μπορεί κανείς να αξιοποιήσει τη συσσωρευμένη εμπειρία άλλων

Πρότυπα Σχεδίασης

...αν υπάρχει κάποιος μεταβατικός χώρος – μία βεράντα με κάλυψη αλλά ανοικτή – αυτό είναι ψυχολογικά στο ενδιάμεσο του μέσα και έξω και κάνει απλούστερη και ευκολότερη την πρόσβαση στην αυλή

με έναρξη το πρότυπο προκύπτουν περαιτέρω δρόμοι για τη βελτίωση της σχεδίασης

Πρότυπα Σχεδίασης

Κάθε πρότυπο περιλαμβάνει:

- το όνομα του προτύπου
- το σκοπό του προτύπου, το πρόβλημα που επιλύει
- τον τρόπο επίλυσης
- τους περιορισμούς που πρέπει να ληφθούν υπόψη

Πρότυπα Σχεδίασης

Αρχές '90:

- Υπάρχουν προβλήματα στο λογισμικό τα οποία επαναλαμβάνονται;
- Υπάρχει η δυνατότητα σχεδίασης λογισμικού με πρότυπα;

Οι Gamma, Helm, Johnson και Vlissides (Gang of Four - GoF) κατέληξαν ότι η απάντηση είναι "**σαφώς ναι**".

Πρότυπα Σχεδίασης

Οι GoF κατήρτισαν έναν κατάλογο με 23 πρότυπα σχεδίασης δίνοντας για το καθένα:

- Όνομα και Κατηγορία
- Σκοπό (μία φράση)
- Συνώνυμα
- Κίνητρο (παράδειγμα προβλήματος)
- Εφαρμοσιμότητα (χρησιμοποιείστε το πρότυπα όταν:…)
- Δομή (διάγραμμα κλάσεων)
- Συμμετέχοντες
- Συνεργασία

Πρότυπα Σχεδίασης

- Συνέπειες (επιτυγχανόμενα πλεονεκτήματα)
- Υλοποίηση (language specific)
- Sample Code
- Γνωστές Χρήσεις (παραδείγματα σε πραγματικά συστήματα)
- Σχετιζόμενα Πρότυπα

Πρότυπα Σχεδίασης

Κατηγορίες

- **Creational**: Ασχολούνται με τη διεργασία της δημιουργίας αντικειμένων
- **Structural**: Διαπραγματεύονται τη σύνθεση κλάσεων/αντικειμένων
- **Behavioral**: Χαρακτηρίζουν τους τρόπους με τους οποίους οι κλάσεις αλληλεπιδρούν και κατανέμουν τις αρμοδιότητες

Πρότυπα Σχεδίασης

Οι GoF προτείνουν τις ακόλουθες στρατηγικές

- Design to interfaces
- Favor composition over inheritance
- Find what varies and encapsulate it

Adapter (Προσαρμογέας)

- Κατηγορία: Structural
- Σκοπός: *Η μετατροπή της διασύνδεσης μιας κλάσης σε μία άλλη που αναμένει το πρόγραμμα πελάτης. Ο προσαρμογέας επιτρέπει τη συνεργασία κλάσεων, η οποία σε διαφορετική περίπτωση θα ήταν αδύνατη λόγω ασύμβατων διασυνδέσεων.*
- Συνώνυμα: Wrapper

Adapter (Προσαρμογέας)

Συχνά ο κώδικας μιας κλάσης προσφέρεται για επαναχρησιμοποίηση, αλλά αυτή δεν είναι δυνατή λόγω του ότι τα προγράμματα που επιθυμούν να χρησιμοποιήσουν τις λειτουργίες της, αναμένουν διαφορετική διασύνδεση.

Έστω ότι μία κλάση Σχεδίασης είναι σε θέση να σχεδιάσει γραμμές, αλλά απαιτεί ως παραμέτρους τις συντεταγμένες στη μορφή (x1, y1, x2, y2) ενώ τα προγράμματα πελάτες είναι σε θέση να παρέχουν τις συντεταγμένες στη μορφή (x1, x2, y1, y2).

Συνήθως τα προγράμματα πελάτες δεν είναι δυνατόν να τροποποιηθούν (καθώς βρίσκονται ήδη εγκατεστημένα).
Η κλάση Σχεδίασης είναι επιθυμητό να χρησιμοποιηθεί χωρίς τροποποίηση (καθώς οποιαδήποτε επέμβαση στον κώδικα μιας μεθόδου είναι δυνατόν να προκαλέσει σφάλματα στις υπόλοιπες μεθόδους).
Εδώ βρίσκει εφαρμογή το πρότυπο "Προσαρμογέας".14

Adapter (Προσαρμογέας)

Παράδειγμα:

Εφαρμογή που χειρίζεται διάφορα σχήματα (σημεία, γραμμές, τετράγωνα) με ενιαίο τρόπο, δηλαδή καλεί λειτουργίες επί των σχημάτων, αδιαφορώντας για το συγκεκριμένο είδος σχήματος.

Τέτοιες λειτουργίες είναι η εμφάνιση του σχήματος, ο καθορισμός του χρώματος, η διαγραφή από την οθόνη, ο καθορισμός θέσης, το γέμισμα με χρώμα κλπ.

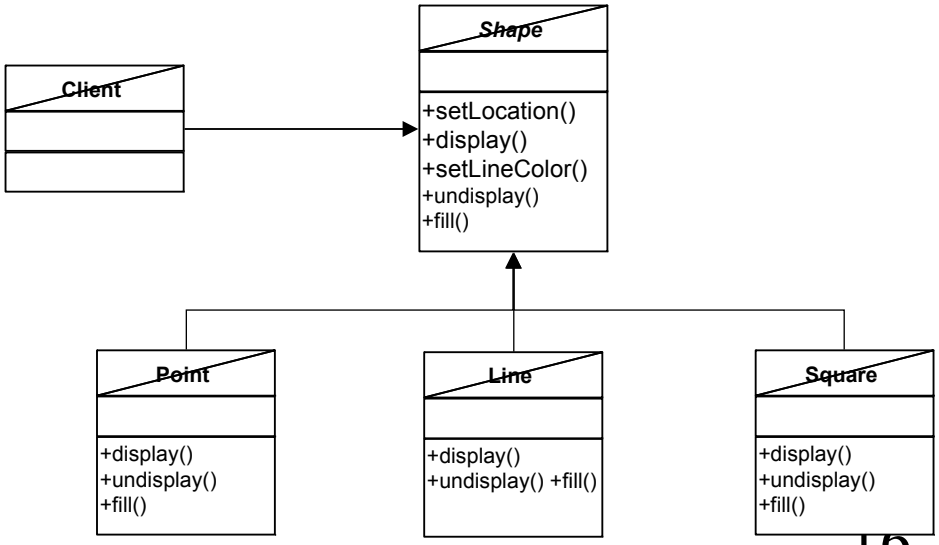
Με άλλα λόγια, όλα τα σχήματα θα πρέπει να ενσωματωθούν σε μία αφηρημένη έννοια σχήματος που θα παρέχει όλες αυτές τις λειτουργίες.

Adapter (Προσαρμογέας)

Προφανής αντιμετώπιση:

Πολυμορφισμός: Μία αφηρημένη κλάση ορίζει όλες τις λειτουργίες της διασύνδεσης, ενώ η υλοποίηση των λειτουργιών είναι διαφορετική για κάθε μία από τις παράγωγες κλάσεις.

Το πρόγραμμα πελάτης είναι σε θέση να χειρίζεται αντικείμενα της αφηρημένης κλάσης διατηρώντας ένα δείκτη προς αυτή, ενώ κατά την εκτέλεση του προγράμματος μεταβιβάζονται ως τιμές στον δείκτη οι διευθύνσεις συγκεκριμένων αντικειμένων των παράγωγων κλάσεων.



Adapter (Προσαρμογέας)

Ζητείται η προσθήκη δυνατότητας σχεδίασης κύκλων: Πρέπει να προστεθεί μία νέα κλάση η οποία θα κληρονομεί από την αφηρημένη κλάση Shape υλοποιώντας την πολυμορφική συμπεριφορά των μεθόδων που δηλώνονται στη διασύνδεση.

Στο σημείο αυτό, θα πρέπει να γραφεί ο κώδικας για τις μεθόδους display, undisplay και fill.

Επειδή η συγγραφή των μεθόδων αυτών είναι αρκετά περίπλοκη, αρχικά πραγματοποιείται αναζήτηση εναλλακτικής λύσης, υπό τη μορφή κάποιας κλάσης που ήδη υλοποιεί κύκλους και είναι διαθέσιμη.

Έστω ότι εντοπίζεται μία τέτοια κλάση με το όνομα OtherCircle, η οποία διαθέτει τις ίδιες λειτουργίες με διαφορετικά ωστόσο ονόματα μεθόδων. Η κλάση OtherCircle δεν μπορεί να ενταχθεί απευθείας στην ιεραρχία των κλάσεων αφενός λόγω της ασυμβατότητας των μεθόδων (ονομάτων και ενδεχομένως παραμέτρων), καθώς και διότι στον ορισμό της δεν κληρονομεί από την κλάση Shape.

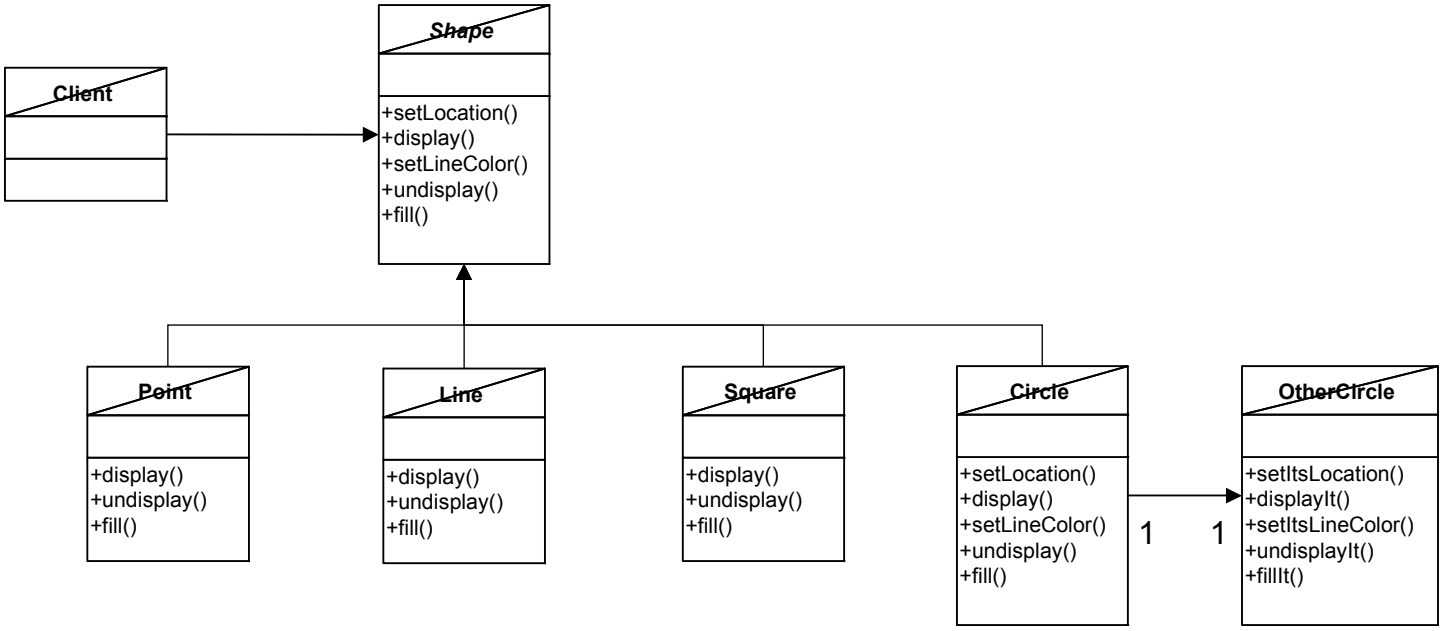
Adapter (Προσαρμογέας)

Η λύση στο σημείο αυτό είναι η δημιουργία μιας νέας κλάσης Circle η οποία όντως κληρονομεί από την Shape (και κατά συνέπεια είναι συμβατή με τις λειτουργίες της διασύνδεσης), και η οποία *περιέχει* ένα αντικείμενο της κλάσης OtherCircle.

Κατά αυτόν τον τρόπο, η κλάση Circle μπορεί να μεταβιβάζει τις αιτήσεις που φθάνουν σε αυτή, στο αντικείμενο OtherCircle που μπορεί να τις ικανοποιήσει. Η υπάρχουσα κλάση OtherCircle επομένως δεν τροποποιείται, αλλά *προσαρμόζεται*.

Διάγραμμα Κλάσεων:

Adapter (Προσαρμογέας)



Adapter (Προσαρμογέας)

Η αφηρημένη κλάση *Shape* ορίζει την αφαίρεση ενός σχήματος ώστε να μπορεί να χρησιμοποιηθεί από προγράμματα-πελάτες. Δύο από τις μεθόδους (`setLocation` και `setLineColor`) είναι υλοποιημένες ενώ οι υπόλοιπες αφήνουν την υλοποίηση στις παράγωγες κλάσεις. (Η κλάση κληρονομεί την JComponent της Java έτσι ώστε να μπορεί να σχεδιαστεί σε ένα παράθυρο).

Shape.java

```
public abstract class Shape extends JComponent {

    public void setLocation(int x, int y) {
        xTopLeft = x;
        yTopLeft = y;
    }

    public void setLineColor(Color c) {
        lineColor = c;
    }

    public abstract void display();
    public abstract void undisplay();
    public abstract void fill(Color c);

    protected int xTopLeft = 0;
    protected int yTopLeft = 0;
    protected Color lineColor;
}
```

Adapter (Προσαρμογέας)

Square.java

```
public class Square extends Shape {

     public Square() {
          rect = new Rectangle(xTopLeft, yTopLeft, 100, 100);
     }

     public void paint(Graphics g) {
          super.paint(g);
          Graphics2D g2 = (Graphics2D)g;
          g2.setColor(lineColor);
          rect.setLocation(xTopLeft, yTopLeft);
          g2.draw(rect);
          if(fillColor !=null) {
            g2.setColor(fillColor);
            g2.fillRect(xTopLeft, yTopLeft, 100, 100);
          } }

     public void fill(Color c) {
          fillColor = c; }

     public void display() {
          this.setVisible(true); }

     public void undisplay() {
          this.setVisible(false); }

     private Rectangle rect;
     private Color fillColor = null;
}
```

Adapter (Προσαρμογέας)

Ένα πρόγραμμα πελάτης προσομοιώνεται από τον ακόλουθο κώδικα. Στο αντικείμενο S1 της κλάσης Square μπορούν να κληθούν όλες οι μέθοδοι που δηλώνονται στη διασύνδεση της Shape.

Client.java

```
public class Client extends JFrame {

    public void draw(Shape componentToDraw)
    {
        getContentPane().add(componentToDraw);
    }

    public static void main(String[] args) {
        Client frame = new Client();

        Square S1 = new Square();
        S1.setLineColor(Color.BLUE);
        S1.setLocation(50, 70);
        frame.draw(S1);
        S1.fill(Color.ORANGE);
        S1.undisplay();
        S1.display();

        frame.setSize(400, 400);
        frame.setVisible(true);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }}
```

Adapter (Προσαρμογέας)

Υποθέτουμε ότι υπάρχει ήδη μία κλάση OtherCircle η οποία διαθέτει υλοποιημένες όλες τις αντίστοιχες μεθόδους, με διαφορετικές όμως υπογραφές, αποκλείοντας την απευθείας χρήση από έναν πελάτη ο οποίος γνωρίζει τη διασύνδεση της Shape. Για παράδειγμα, ένα πρόγραμμα πελάτης, μπορεί να σαρώνει μία λίστα από σχήματα και να τα σχεδιάζει, καλώντας τις ίδιες μεθόδους (αυτές που δηλώνονται στη Shape) για όλα τα σχήματα. Ο κώδικας της OtherCircle φαίνεται παρακάτω:

OtherCircle.java

```
import javax.swing.*;
import java.awt.*;

public class OtherCircle extends JComponent {

    public void paint(Graphics g) {
        super.paint(g);
        Graphics2D g2 = (Graphics2D)g;
        g2.setColor(lineColor);
        g2.drawOval(xUpperLeft, yUpperLeft, 100, 100);
        if(fillColor != null) {
          g2.setColor(fillColor);
          g2.fillOval(xUpperLeft, yUpperLeft, 100, 100);
        }
    }

//Αυτή η μέθοδος λαμβάνει τα ορίσματα με αντίστροφη σειρά
    public void setItsLocation(int y, int x) {
        xUpperLeft = x;
        yUpperLeft = y;
    }
```

OtherCircle.java

```
//Διαφορετικά ονόματα μεθόδων και ιδιοτήτων
        public void setItsLineColor(Color c) {
                lineColor = c;
        }

        public void fillIt(Color c) {
                fillColor = c;
        }

        public void displayIt() {
                this.setVisible(true);
        }

        public void undisplayIt() {
                this.setVisible(false);
        }

        private Color fillColor = null;
        private int xUpperLeft = 0;
        private int yUpperLeft = 0;
        private Color lineColor;
}
```

Adapter (Προσαρμογέας)

Για την προσαρμογή μιας τέτοιας κλάσης στην υπάρχουσα διασύνδεση, δημιουργείται μία κλάση "Προσαρμογέας"

- η οποία κληρονομεί τη Shape και

-κατά τη δημιουργία της δημιουργεί με τη σειρά της ένα αντικείμενο της προσαρμοζόμενης κλάσης OtherShape.

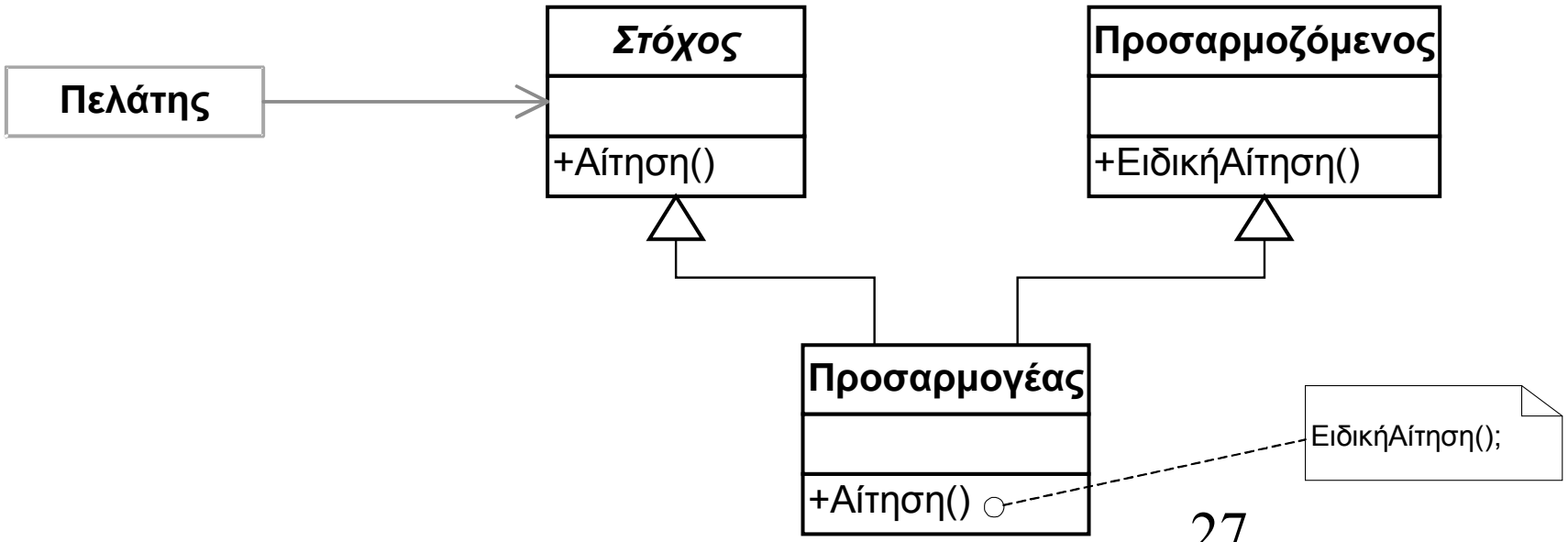
Η κλάση Circle που έχει το ρόλο του προσαρμογέα, υλοποιεί όλες τις μεθόδους αποστέλλοντας κατάλληλα μηνύματα στο προσαρμοζόμενο αντικείμενο.

Adapter (Προσαρμογέας)

Γενική Δομή – Εφαρμογή

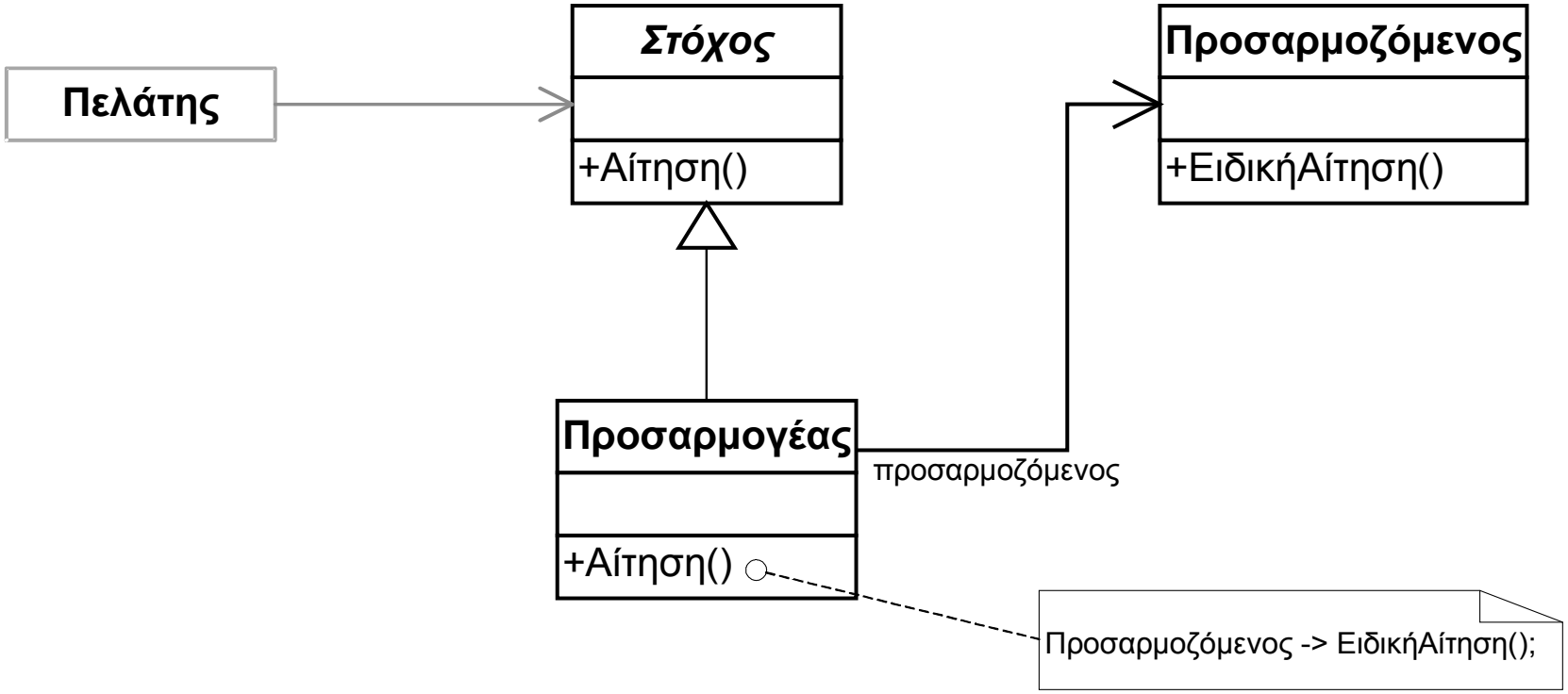
• θέλετε να χρησιμοποιήσετε μία υπάρχουσα κλάση, αλλά η διασύνδεσή της δεν συμβαδίζει με τις ανάγκες σας.

• Ένας προσαρμογέας κλάσης (class adapter) χρησιμοποιεί πολλαπλή κληρονομικότητα για να προσαρμόσει μία διασύνδεση σε μία άλλη.



Adapter (Προσαρμογέας)

• Ένας προσαρμογέας αντικειμένου (object adapter) βασίζεται στη σύνθεση αντικειμένων και στη διαβίβαση μηνυμάτων (delegation).



Composite (Σύνθετο)

- Κατηγορία: Structural
- Σκοπός: *Η σύνθεση αντικειμένων σε δενδροειδείς δομές για την αναπαράσταση ιεραρχιών τμήματος-όλου. Το πρότυπο σχεδίασης "Σύνθετο" επιτρέπει στα προγράμματα πελάτες να διαχειρίζονται με ενιαίο τρόπο τόσο τα ανεξάρτητα αντικείμενα όσο και συνθέσεις αντικειμένων*
- Συνώνυμα: -

Composite (Σύνθετο)

Πολύ συχνά, σε μία εφαρμογή, εκτός από μεμονωμένα αντικείμενα (π.χ. Τροχός, Μηχανή, Κάθισμα), υφίστανται και σύνθετα αντικείμενα που περιέχουν ή περιλαμβάνουν άλλα αντικείμενα (π.χ. Αυτοκίνητο).

Συνήθης αντιμετώπιση: χρήση μιας σχέσης περιεκτικότητας μεταξύ της κλάσης που αντιπροσωπεύει το όλον (περικλείουσα κλάση) και των κλάσεων που αντιπροσωπεύουν τα τμήματα.

Μειονέκτημα: Δεν επιτρέπει τον ομοιόμορφο χειρισμό των αντικειμένων από ένα πρόγραμμα πελάτη.

Composite (Σύνθετο)

Για παράδειγμα, μία άλλη εφαρμογή, θα πρέπει να διατηρεί δείκτες και προς αντικείμενα τύπου Τροχός, Μηχανή, Κάθισμα, αλλά και δείκτες προς αντικείμενα τύπου Αυτοκίνητο.

Λαμβάνοντας υπόψη τους πρωταρχικούς στόχους του αντικειμενοστραφούς προγραμματισμού, αν προστεθεί στο σύστημα μία νέα σύνθετη κλάση (π.χ. Λεωφορείο), τότε ο κώδικας του προγράμματος πελάτη, θα πρέπει να τροποποιηθεί για να είναι δυνατός ο χειρισμός των νέων αντικειμένων τύπου Λεωφορείο.

Η αντιμετώπιση αυτού του προβλήματος, επιτυγχάνεται με κομψό τρόπο με το πρότυπο σχεδίασης "Σύνθετο".

Composite (Σύνθετο)

Παράδειγμα:

Οι εφαρμογές που σχεδιάζουν πολύπλοκα ψηφιακά κυκλώματα VLSI αντιμετωπίζουν οποιαδήποτε οντότητα ως αντικείμενο.

Κάθε αντικείμενο έχει ορισμένες λειτουργίες, μεταξύ αυτών η σχεδίαση του.

Ορισμένα από τα αντικείμενα είναι πρωταρχικά και δεν αναλύονται περαιτέρω (π.χ. λογικές πύλες AND, OR, NOT), ενώ άλλα αντικείμενα είναι σύνθετα και αποτελούνται από πρωταρχικές πύλες (π.χ. ένας πλήρης αθροιστής – Full Adder).

Composite (Σύνθετο)

Παράδειγμα:

Ο χρήστης είναι σε θέση να δημιουργήσει οποιαδήποτε σύνθετη οντότητα και να την προσθέσει στην εφαρμογή. Η σχεδίαση ενός σύνθετου αντικειμένου ουσιαστικά συνίσταται στη σχεδίαση των επί μέρους τμημάτων του.

Για το λόγο αυτό, είναι επιθυμητή η ενιαία αντιμετώπιση όλων των αντικειμένων.

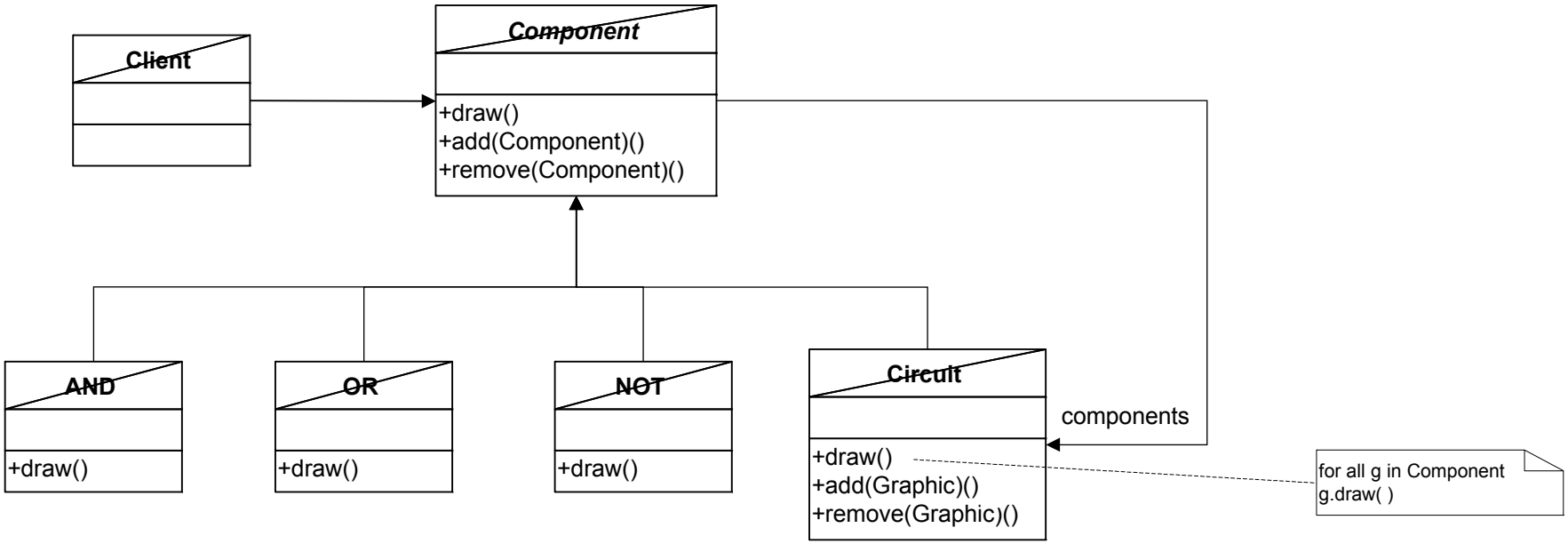
Το πρότυπο σχεδίασης "Σύνθετο", επιτρέπει τον αναδρομικό ορισμό περιεκτικότητας, ώστε οι πελάτες να μην αντιλαμβάνονται τη διαφορά μεταξύ πρωταρχικών και σύνθετων αντικειμένων.

Composite (Σύνθετο)

Το σημείο κλειδί στο πρότυπο σχεδίασης "Σύνθετο" είναι η ύπαρξη μιας αφηρημένης κλάσης που αναπαριστά τόσο πρωταρχικές κλάσεις όσο και περικλείουσες κλάσεις.

Για τη συγκεκριμένη εφαρμογή, η αφηρημένη κλάση είναι η κλάση Εξάρτημα. Θα πρέπει να σημειωθεί, ότι η κλάση Εξάρτημα που εισάγεται από το πρότυπο, δεν έχει αντιστοιχία με οποιαδήποτε φυσική έννοια του πραγματικού κόσμου.

Composite (Σύνθετο)



Composite (Σύνθετο)

Θέτοντας τις λειτουργίες χειρισμού των αντικειμένων που εμπεριέχονται σε μία περικλείουσα κλάση (add, remove) στην κλάση Component, όλα τα αντικείμενα αντιμετωπίζονται με τον ίδιο ακριβώς τρόπο, εξασφαλίζοντας διαφάνεια. Ωστόσο δημιουργείται το εξής πρόβλημα: Για τις κλάσεις φύλλα (όπως οι AND, OR, NOT) είναι δυνατόν να κληθούν οι λειτουργίες add, remove οι οποίες δεν έχουν νόημα.

Εναλλακτικά, οι λειτουργίες αυτές είναι δυνατόν να δηλωθούν στην κλάση Circuit, ώστε οποιαδήποτε απόπειρα να προστεθούν ή να διαγραφούν αντικείμενα σε αντικείμενα τύπου AND, OR ή NOT να ανιχνευθεί κατά τη μεταγλώττιση, εξασφαλίζοντας ασφάλεια. Ωστόσο, στην περίπτωση αυτή, οι πρωταρχικές κλάσεις και οι σύνθετες κλάσεις έχουν διαφορετική διασύνδεση

Composite (Σύνθετο)

```
class Component {
public:
    Component(string text);
    virtual void draw() = 0;   //αμιγώς υπερβατική -> //δεν
                               υλοποιούνται αντικείμενα

    virtual void add(Component*) {};
    virtual void remove(Component*) {};
protected:
    string identifier;
};

Component::Component(string text)
{
    identifier = text;
}
```

Composite (Σύνθετο)

Μία πρωταρχική κλάση, όπως μία κλάση πύλης AND, δηλώνεται ως παράγωγη της κλάσης Component:

```
class AND : public Component {
public:
    AND(string text);
    virtual void draw();
};

void AND::draw()
{
    cout << "AND " << identifier
         << " is being drawn" << endl;
}
```

Composite (Σύνθετο)

Η κλάση Circuit είναι μία περικλείουσα κλάση η οποία είναι δυνατόν να περιέχει αντικείμενα:

- πρωταρχικών κλάσεων (π.χ. ένα κύκλωμα που αποτελείται από πύλες),
- άλλων σύνθετων κλάσεων (π.χ. ένα κύκλωμα που αποτελείται από άλλα κυκλώματα)
- αντικείμενα και των δύο κατηγοριών (ένα κύκλωμα που αποτελείται από πύλες και άλλα κυκλώματα).

Για το λόγο αυτό, μία από τις ιδιότητες υλοποιεί έναν περιέχοντα (container), όπως ένα διάνυσμα (vector), ουρά (queue), διπλά συνδεδεμένη λίστα (list), στοίβα (stack) κ.ο.κ, που περιλαμβάνονται στην πρότυπη βιβλιοθήκη της C++ (STL):

Composite (Σύνθετο)

```
class Circuit : public Component {
public:
    Circuit(string text);
    virtual void draw();

    virtual void add(Component*);
    virtual void remove(Component*);
private:
    vector<Component*> components;
};
```

Composite (Σύνθετο)

Η λειτουργία `draw` χρησιμοποιεί έναν επαναλήπτη (Iterator) για να διατρέξει όλα τα στοιχεία του διανύσματος `components`, και να καλέσει μέσω αυτού τη λειτουργία `draw` του κάθε αντικειμένου που περιλαμβάνεται:

```
void Circuit::draw()
{
    cout <<"Circuit "<<identifier<< " is being drawn" << endl;

    std::vector<Component*>::iterator i;

    for(i=components.begin(); i!=components.end(); i++)
        (*i)->draw();

}
```

Composite (Σύνθετο)

Οι λειτουργίες `add` και `remove` εισάγουν και διαγράφουν αντικείμενα τύπου `Component` (**άρα οτιδήποτε**) από το διάνυσμα που είναι αποθηκευμένο στο μέλος δεδομένων `components`:

```
void Circuit::add(Component* comp)
{
    components.push_back(comp);
}

void Circuit::remove(Component* comp)
{
    vector<Component*>::iterator i;

    for(i=components.begin(); i!=components.end(); i++)
        if((*i) == comp)
            break;

    components.erase(i);
}
```

Composite (Σύνθετο)

Με τη χρήση του ανωτέρου προτύπου είναι πλέον δυνατή η δημιουργία οποιουδήποτε πρωταρχικού ή σύνθετου αντικειμένου και ο χειρισμός τους με ενιαίο τρόπο:

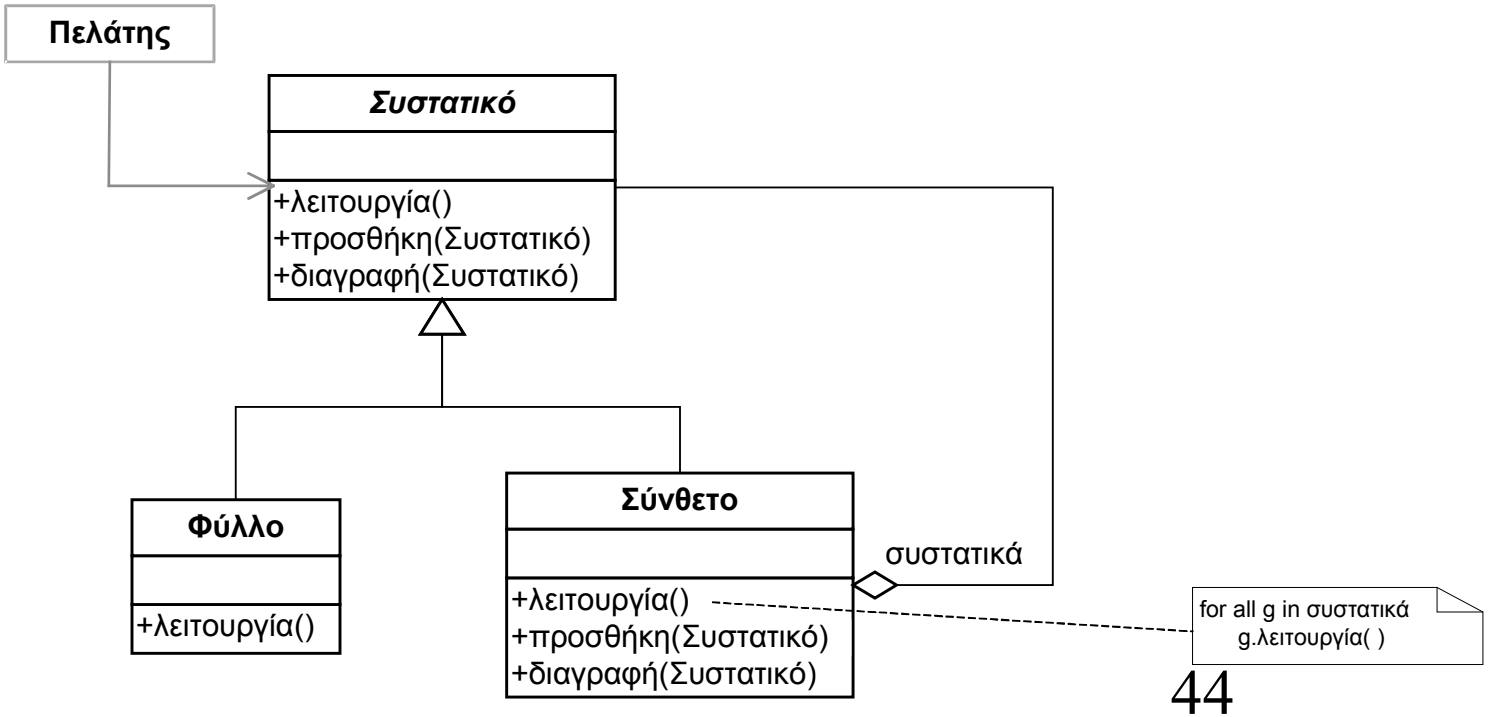
```
int main() {
    AND G1("Gate1");
    AND G2("Gate2");

    Circuit C1("Circuit1");
    C1.add(&G1);
    C1.add(&G2);

    AND G3("Gate3");

    Circuit C2("Circuit2");
    C2.add(&G3);          //προσθήκη απλού στοιχείου
    C2.add(&C1);          //προσθήκη σύνθετου στοιχείου

    G1.draw();            //κοινή διασύνδεση
    C2.draw();

    return 0; }
```

Composite (Σύνθετο)

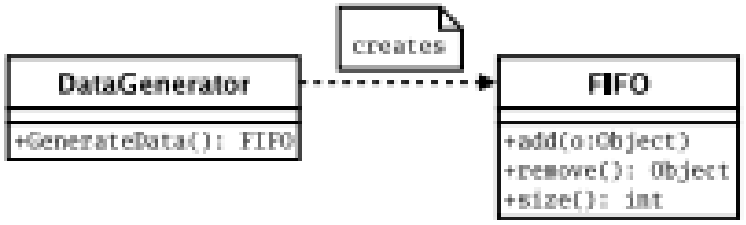
Γενική Δομή – Εφαρμογή

Το πρότυπο σχεδίασης "Σύνθετο" χρησιμοποιείται όταν:

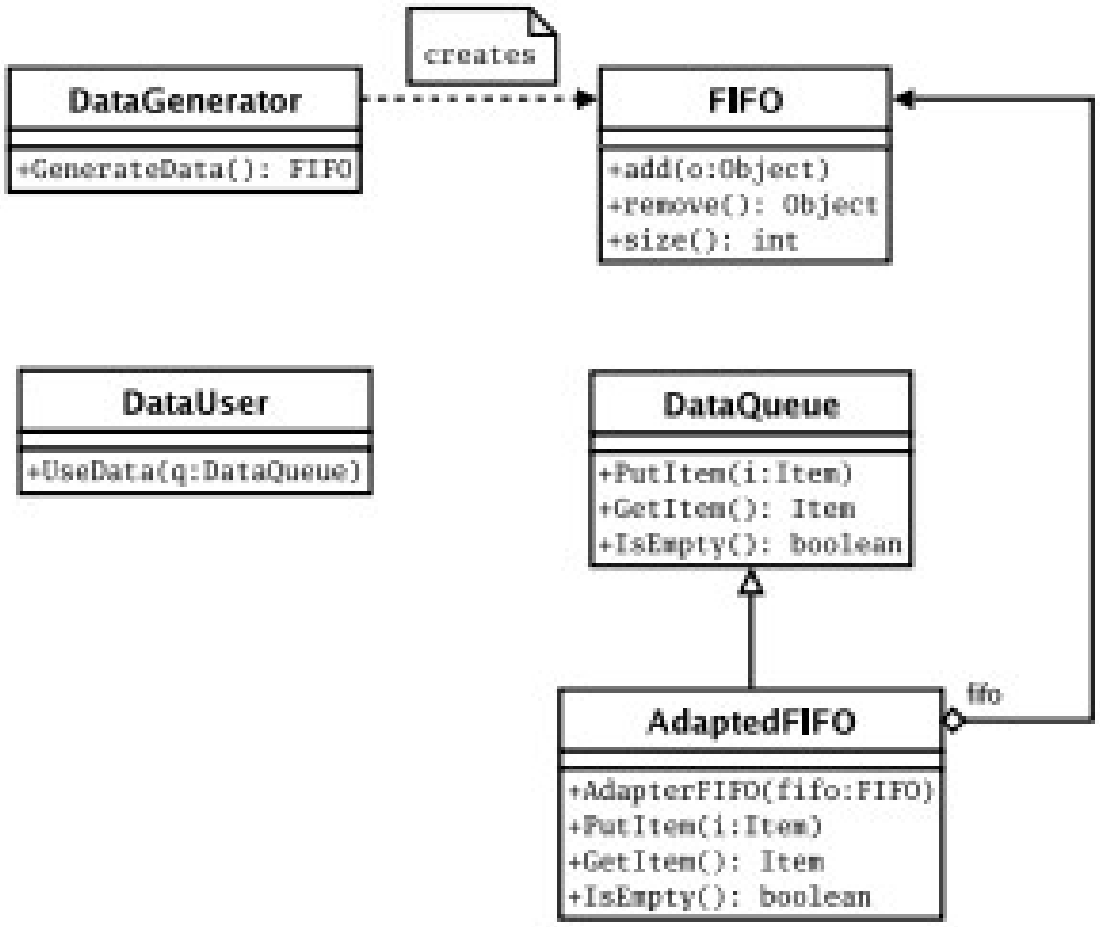
• θέλετε τα προγράμματα πελάτες να μπορούν να χειρίζονται μεμονωμένα αντικείμενα και σύνθετα αντικείμενα (συλλογές από άλλα αντικείμενα) με τον ίδιο τρόπο



- **Situation: Two parts of a program are already implemented; a class that generates data and another class that uses data.**



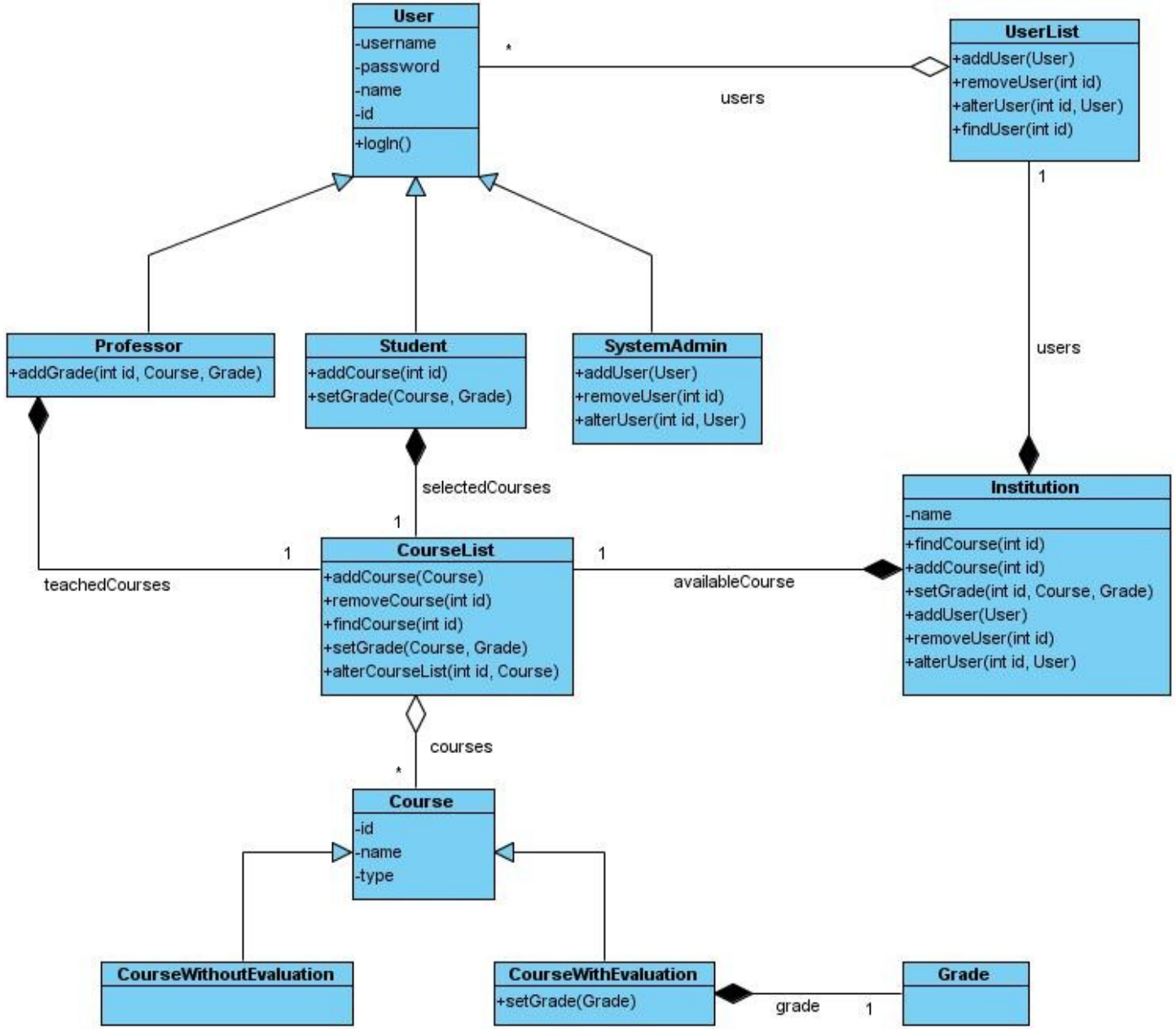
- Problem: FIFO and DataQueue have incompatible interfaces!
- But they do the same thing → adapt!

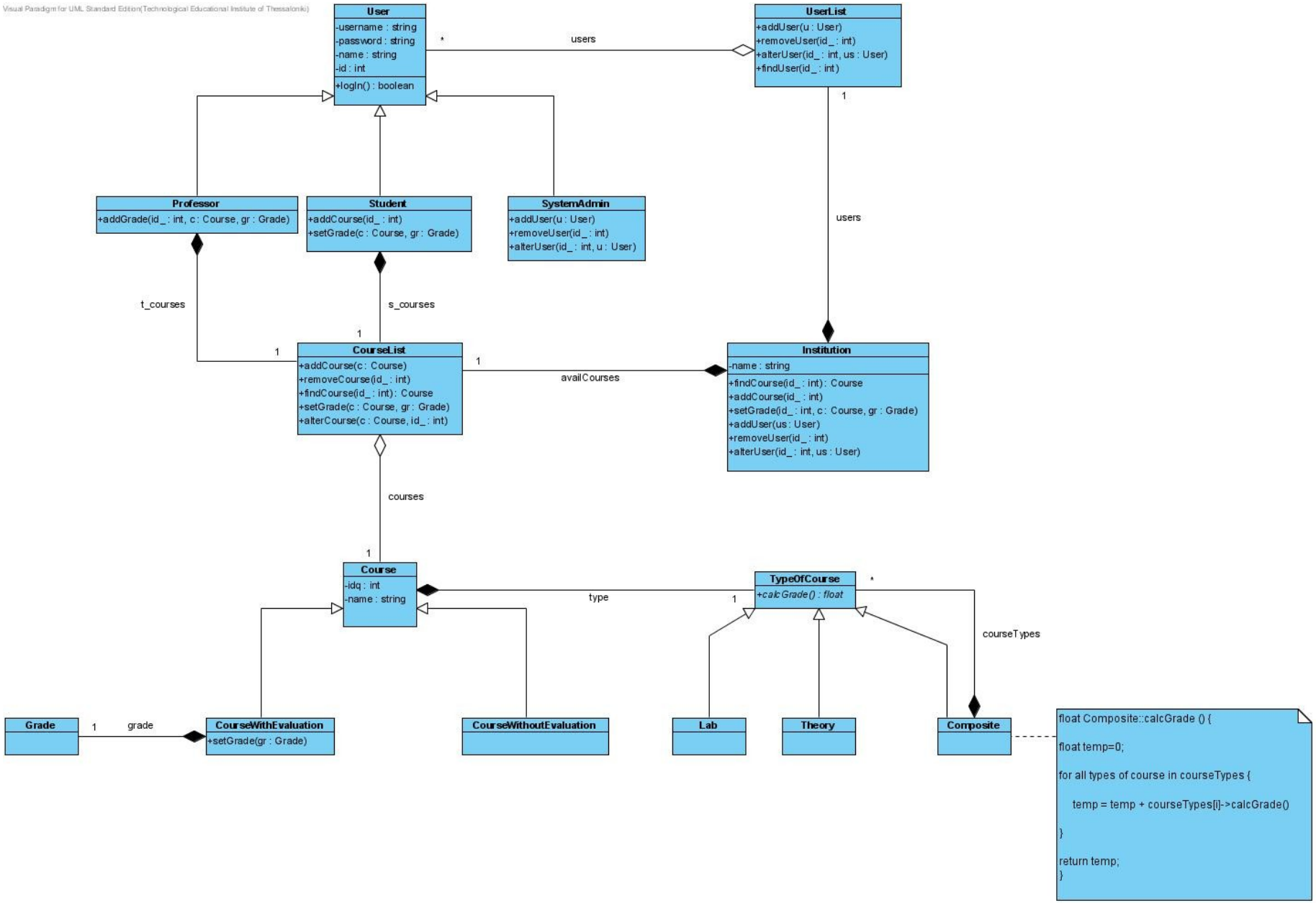

creates
DataGenerator
+GenerateData(): FIFO
FIFO
+add(o:Object)
+remove(): Object
+size(): int
DataUser
+UseData(q:DataQueue)
DataQueue
+PutItem(i:Item)
+GetItem(): Item
+IsEmpty(): boolean
fifo
AdaptedFIFO
+AdapterFIFO(fifo:FIFO)
+PutItem(i:Item)
+GetItem(): Item
+IsEmpty(): boolean

```
public class AdapterFIFO extends DataQueue {
  public AdapterFIFO(FIFO fifo){
    this.fifo=fifo;
  }
  public void putItem(Item i){
    fifo.add( (Object) i);
  }
  public Item getItem(){
/* Assume we can type cast */
    return (Item) fifo.remove();
  }
  public boolean isEmpty(){
    return ( fifo.getSize() == 0 );
  }
}
```

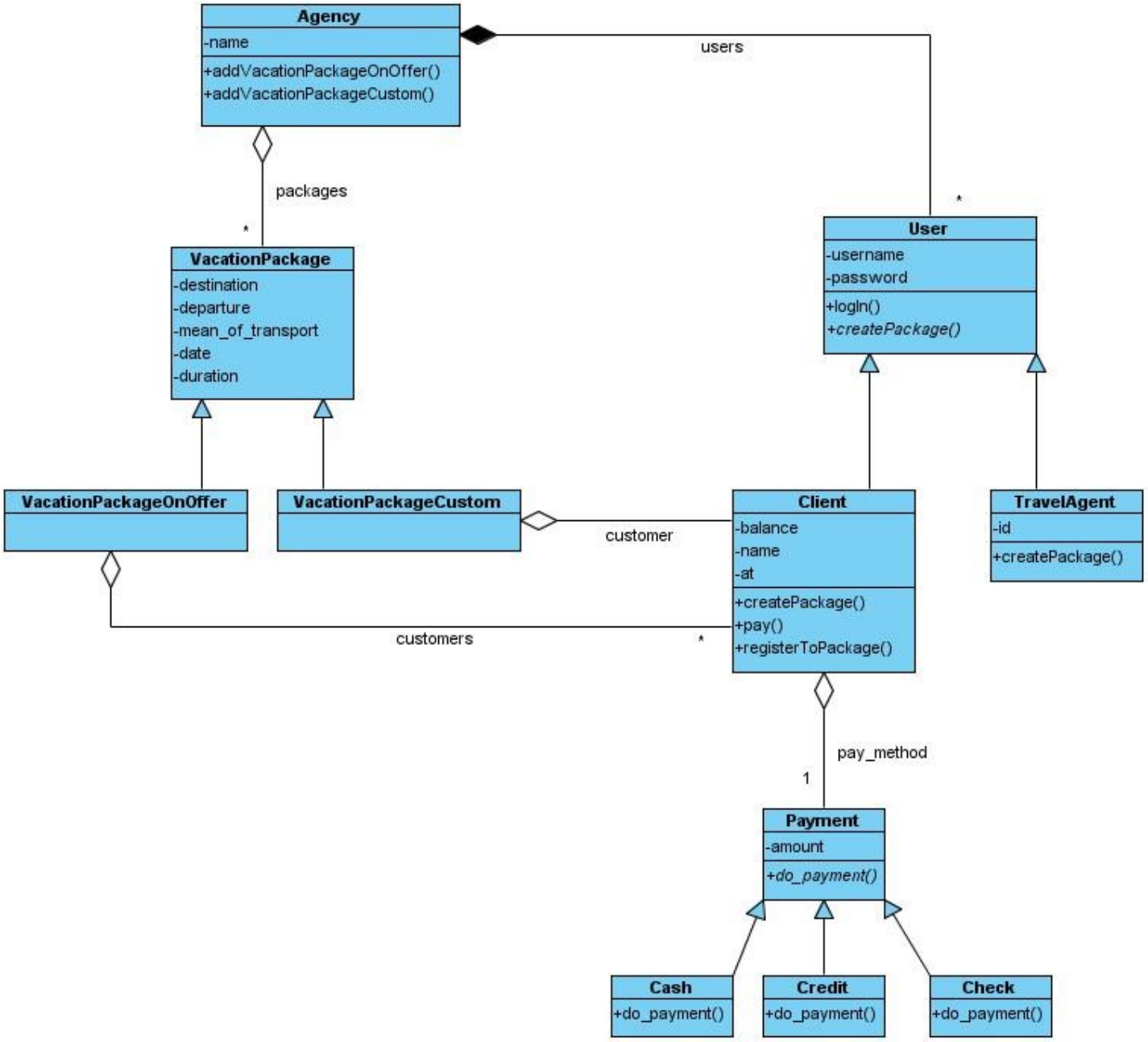
- Έστω ένα σύστημα διαχείρισης βαθμολογιών, όπως παρουσιάζεται από τον κώδικα και το διάγραμμα κλάσεων. Σε αυτό το σύστημα προσθέστε τις παρακάτω λειτουργίες:
  - Ένα μάθημα μπορεί να είναι εργαστηριακό, θεωρίας ή συνδυασμό των παραπάνω, ο βαθμός προκύπτει με διαφορετικό τρόπο για το καθένα.

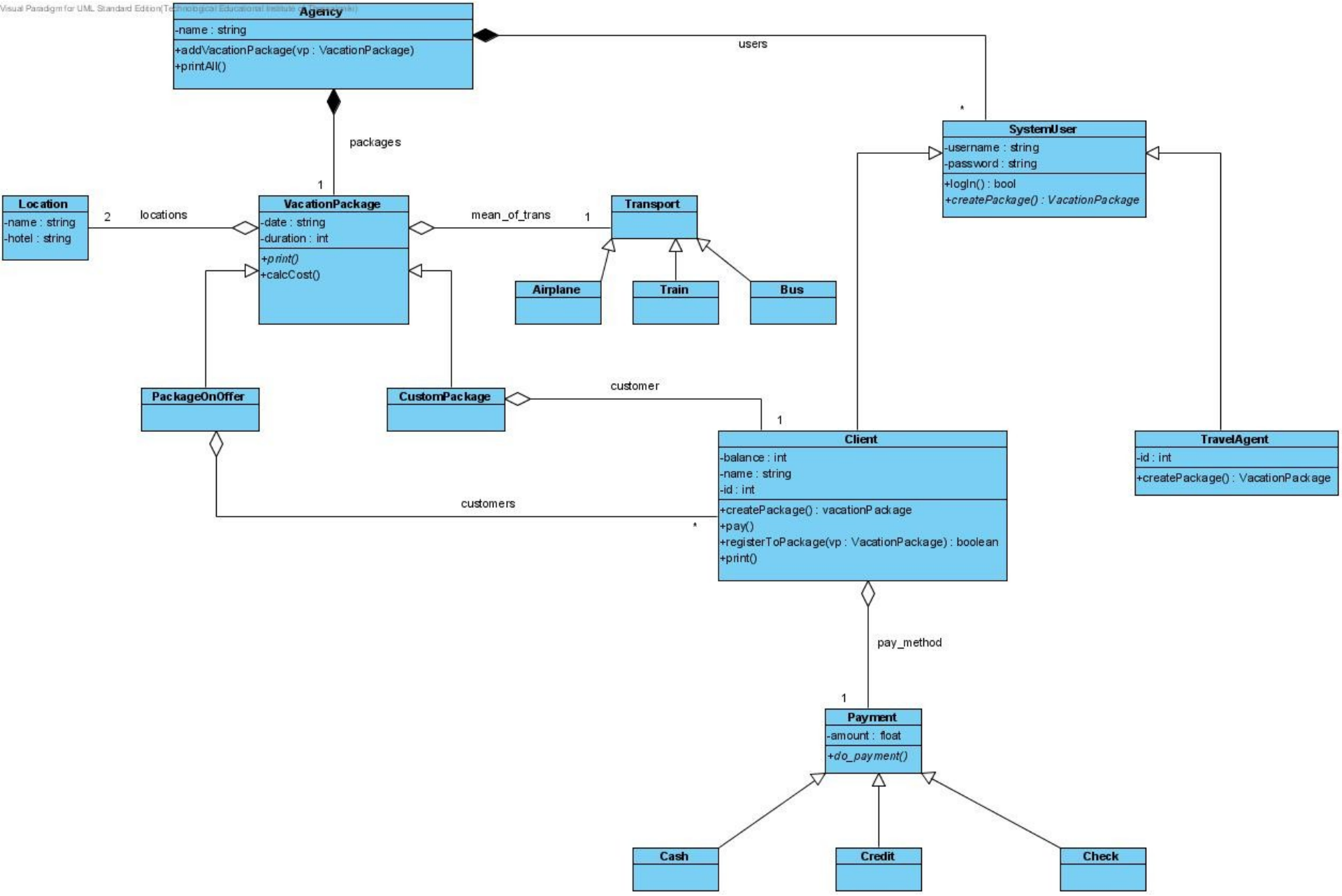
```
void Professor::addGrade(int student_id, Course c, Grade g) {

   // Let courseId be the id of the course under study
   c = teachedCourses.findCourse(courseId)
   // Let grade_ be the grade of the corresponding student
   g = Grade (grade_);
   institution.setGrade(student_id, c, g);
}

void Institution::setGrade(int student_id, Course c, Grade g) {
   Student s = users.findUser(student_id);
   bool success = s.setGrade(c,g);
   if (success==true)
      users.alterUser(student_id,s);
   else; // message
}

bool Student::setGrade(Course c, Grade g) {
   bool s = courses.findCourse(c.getId());
   if (s == false)
      return false;
   else {
      courses.setGrade(c,g);
      return true;
   }
}

void courseList::setGrade(Course c, Grade g) {
   Course c_ = new CourseWithEvaluation(c);
   c_.setGrade(g);
   alterCourseList(c.getId(), c_);
}
```


User
-username
-password
-name
-id
+logIn()
*
users
UserList
+addUser(User)
+removeUser(int id)
+alterUser(int id, User)
+findUser(int id)
1
users
Professor
+addGrade(int id, Course, Grade)
Student
+addCourse(int id)
+setGrade(Course, Grade)
SystemAdmin
+addUser(User)
+removeUser(int id)
+alterUser(int id, User)
selectedCourses
1
Institution
-name
+findCourse(int id)
+addCourse(int id)
+setGrade(int id, Course, Grade)
+addUser(User)
+removeUser(int id)
+alterUser(int id, User)
1
teachedCourses
CourseList
+addCourse(Course)
+removeCourse(int id)
+findCourse(int id)
+setGrade(Course, Grade)
+alterCourseList(int id, Course)
1
availableCourse
courses
*
Course
-id
-name
-type
CourseWithoutEvaluation
CourseWithEvaluation
+setGrade(Grade)
grade
1
Grade


Visual Paradigm for UML Standard Edition(Technological Educational Institute of Thessaloniki)
User
-username : string
-password : string
-name : string
-id : int
+logIn() : boolean
*
users
UserList
+addUser(u : User)
+removeUser(id_ : int)
+alterUser(id_ : int, us : User)
+findUser(id_ : int)
1
users
Professor
+addGrade(id_ : int, c : Course, gr : Grade)
Student
+addCourse(id_ : int)
+setGrade(c : Course, gr : Grade)
SystemAdmin
+addUser(u : User)
+removeUser(id_ : int)
+alterUser(id_ : int, u : User)
t_courses
s_courses
1
1
CourseList
+addCourse(c : Course)
+removeCourse(id_ : int)
+findCourse(id_ : int) : Course
+setGrade(c : Course, gr : Grade)
+alterCourse(c : Course, id_ : int)
1
availCourses
Institution
-name : string
+findCourse(id_ : int) : Course
+addCourse(id_ : int)
+setGrade(id_ : int, c : Course, gr : Grade)
+addUser(us : User)
+removeUser(id_ : int)
+alterUser(id_ : int, us : User)
courses
1
Course
-idq : int
-name : string
type
1
TypeOfCourse
+calcGrade() : float
*
courseTypes
Grade
1
grade
CourseWithEvaluation
+setGrade(gr : Grade)
CourseWithoutEvaluation
Lab
Theory
Composite
float Composite::calcGrade () {
float temp=0;
for all types of course in courseTypes {
temp = temp + courseTypes[i]->calcGrade()
}
return temp;
}

- Έστω ένα σύστημα ταξιδιωτικού γραφείου, όπως παρουσιάζεται από το παρακάτω διάγραμμα κλάσεων. Σε αυτό το σύστημα προσθέστε τις παρακάτω λειτουργίες:
  - Ο Ταξιδιωτικός Πράκτορας θα μπορεί να παίρνει από το σύστημα εκτυπώσεις σχετικά με τους εγγεγραμμένους πελάτες σε κάθε πακέτο και λίστα με τις προσφορές που έχουν δημιουργηθεί μεμονωμένα από τους πελάτες.
  - Λαμβάνοντας υπ' όψη ότι οι ιδιότητες departure και destination περιλαμβάνουν πληροφορίες της ίδιας κατηγορίας, επιθυμητή είναι η δημιουργία μιας νέας κλάσης Location (τοποθεσία).
  - Ο ορισμός του μέσου μεταφοράς που χρησιμοποιήθηκε θα πρέπει να γίνετε μέσω των κλάσεων Transport, Airplane, Train, Ship, Bus.
  - Θα πρέπει στο νέο σύστημα να υπάρχουν πληροφορίες σχετικά με τα ξενοδοχεία στον τόπο παραμονής και αυτόματος υπολογισμός αξίας στη κλάση VacationPackage


Agency
-name
+addVacationPackageOnOffer()
+addVacationPackageCustom()
users
packages
*
*
User
-username
-password
+logIn()
+createPackage()
VacationPackage
-destination
-departure
-mean_of_transport
-date
-duration
VacationPackageOnOffer
VacationPackageCustom
customer
Client
-balance
-name
-at
+createPackage()
+pay()
+registerToPackage()
TravelAgent
-id
+createPackage()
customers
*
pay_method
1
Payment
-amount
+do_payment()
Cash
+do_payment()
Credit
+do_payment()
Check
+do_payment()


Agency
-name : string
+addVacationPackage(vp : VacationPackage)
+printAll()
users
*
packages
1
SystemUser
-username : string
-password : string
+logIn() : bool
+createPackage() : VacationPackage
Location
-name : string
-hotel : string
2
locations
VacationPackage
-date : string
-duration : int
+print()
+calcCost()
mean_of_trans
1
Transport
Airplane
Train
Bus
PackageOnOffer
CustomPackage
customer
1
Client
-balance : int
-name : string
-id : int
+createPackage() : vacationPackage
+pay()
+registerToPackage(vp : VacationPackage) : boolean
+print()
TravelAgent
-id : int
+createPackage() : VacationPackage
customers
*
pay_method
1
Payment
-amount : float
+do_payment()
Cash
Credit
Check