Πανεπιστήμιο Δυτικής Αττικής
Τμήμα Μηχανικών Πληροφορικής και Υπολογιστών

Αντικειμενοστρεφής Προγραμματισμός

Εργαστήριο 6

Λίγα λόγια για την UML

Η UML (Unified Modeling Language) είναι μια γλώσσα μέσω της οποία μπορούμε να μοντελοποιήσουμε και να παραστήσουμε συμβολικά το σύστημα που θέλουμε να αναπτύξουμε. Η UML πλέον είναι η πρότυπη γλώσσα μοντελοποίησης στη μηχανική λογισμικού. Χρησιμοποιείται για τη γραφική απεικόνιση, προσδιορισμό, κατασκευή και τεκμηρίωση των στοιχείων ενός συστήματος λογισμικού.

Το βασικό τμήμα της UML που θα μας φανεί χρήσιμο είναι τα διαγράμματα κλάσεων. Τα διαγράμματα κλάσεων της UML χρησιμοποιούν γεωμετρικά σχήματα ως συμβολισμούς για τα αντικείμενα, τις κλάσεις και τις διασυνδέσεις, ενώ διαφόρων τύπων γραμμές χρησιμοποιούνται για να συνδέουν αυτά τα σχήματα και να υποδηλώνουν έτσι τον τρόπο που κληρονομούν, συνεργάζονται ή εξαρτώνται μεταξύ τους. Τα αντικείμενα της ίδιας κλάσης αναπαριστώνται με ένα μόνο γεωμετρικό σχήμα.

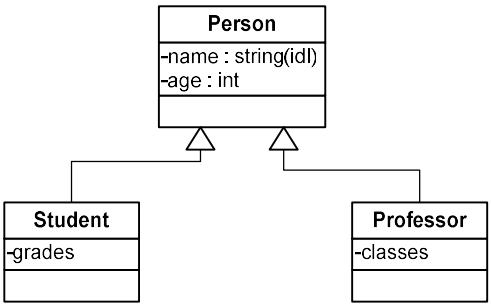
Inheritance, Composition και Aggregation (Κληρονομικότητα, Σύνθεση, Συνάρθοιση)

Ένα σύνηθες πρόβλημα που υπάρχει κατά τον σχεδιασμό και την ανάλυση ενός προγράμματος, και πιο συγκεκριμένα κατά την μοντελοποίηση των αντικειμένων που αυτό περιέχει, είναι η δυσκολία διάκρισης μεταξύ των διαφορετικών συσχετίσεων μεταξύ των αντικειμένων.

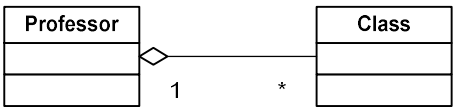
Υπάρχουν, μεταξύ άλλων, τρεις βασικές σχέσεις μεταξύ των αντικειμένων:

- Η κληρονομικότητα (inheritance) όπου ένα αντικείμενο κληρονομεί από ένα άλλο
- Η σύνθεση (composition) όπου ένα αντικείμενο περιέχει εσωτερικά άλλα αντικείμενα. Τα αντικείμενα αυτά είναι αποκλειστικά δικά του και δεν έχουν νόημα ύπαρξης έξω από το αντικείμενο αυτό.
- Η συνάθροιση (aggregation), όπου ένα αντικείμενο περιέχει αναφορές σε άλλα αντικείμενα. Τα αντικείμενα αυτά δεν είναι αποκλειστικά δικά του (μπορεί δηλαδή να τα μοιράζεται με άλλα αντικείμενα) και επιπλέον έχουν νόημα ύπαρξης ΚΑΙ έξω από το αντικείμενο αυτό.

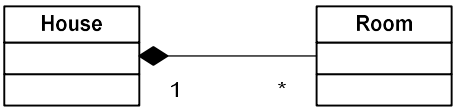
Η κληρονομικότητα πρέπει να χρησιμοποιηθεί όταν θέλουμε να προσθέσουμε νέες λειτουργίες σε μια κλάση, χωρίς να χρειαστεί να την αλλάξουμε: απλά δημιουργούμε μια νέα κλάση που κληρονομεί και εξειδικεύει την μητρική της. Στο ακόλουθο παράδειγμα, που φαίνεται σε UML, οι κλάσεις Student και Professor εξειδικεύουν την γενικότερη κλάση Person.



Στο ακόλουθο παράδειγμα, ένα αντικείμενο Professor μπορεί να περιέχει ένα πίνακα με αντικείμενα της κλάσης Class (τα μαθήματα που διδάσκει). Η σχέση αυτή μεταξύ των δύο κλάσεων λέγεται aggregation και παριστάνεται με στο διάγραμμα UML με ένα άσπρο διαμάντι.



Ένα παράδειγμα σύνθεσης (composition) είναι ένα σπίτι και τα δωμάτια που το αποτελούν. Αν το σπίτι γκρεμιστεί, τότε τα δωμάτια θα καταστραφούν και αυτά, οπότε δεν έχουν λόγο ύπαρξης εκτός του σπιτιού. Η σχέση αυτή σχεδιάζεται στην UML με ένα μαύρο διαμάντι.



Ένας τρόπος (από τους πολλούς) για να αντιμετωπίσουμε προγραμματιστικά την σύνθεση είναι ο εξής:

```
class Room
{
	string name;
	float area;
	//other properties and methods follow….
};

class House
{
private:
	Room rooms[10];
};
```

δηλαδή, με την δημιουργία ενός αντικειμένου House δημιουργούνται και τα αντίστοιχα αντικείμενα τύπου Room. Αντίστοιχα, κατά την καταστροφή του House, όλα τα αντικείμενα τύπου Room θα καταστραφούν επίσης.

Ένας από τους τρόπους αντιμετώπισης της συνάθροισης είναι ο ακόλουθος:

```
class Lecture
{
	string title;
	string description;
};

class Professor
{
```

```
private:
	Lecture* lectures[10];
	int top; //number of lectures provided by this Professor

public:
	Lecture(): top(0) {  }

	void addLecture(Lecture *lec)
	{
		if(top < 10)
			lectures[top++] = lec;
	}
};

void main()
{
	Professor prof("myname");
	Lecture lec001("Math1");

	prof.addLecture(&lec001);
}
```

Εδώ, ο Professor περιέχει όχι αντικείμενα αλλά pointers σε Lectures. Έτσι όταν το αντικείμενο Professor καταστραφεί, τότε δεν θα χαθούν και τα αντίστοιχα Lectures.

Ένα πολύ συχνό λάθος κατά την μοντελοποίηση του συστήματος που θέλουμε να αναπτύξουμε είναι το μπέρδεμα μεταξύ των τριών αυτών σχέσεων, με αποτέλεσμα οι κλάσεις που δημιουργούνται να μην είναι λογικά σωστές. Για παράδειγμα, ένα σοβαρό λάθος που γίνεται συχνά από νέους προγραμματιστές είναι κάποιος να θεωρήσει ότι ο Professor (στο παραπάνω παράδειγμα) θα πρέπει να κληρονομεί και από την Class (εκτός από την Person).

Παράδειγμα Aggregation

Στο παρακάτω παράδειγμα εμφανίζεται η σχέση μεταξύ Φοιτητή και Καθηγητή. Το αντικείμενο του Φοιτητή περιέχει, μεταξύ άλλων, και ένα pointer σε ένα ήδη υπάρχον αντικείμενο Καθηγητή, ο οποίος είναι ο επιβλέπων του (π.χ. στην πτυχιακή του εργασία). Όταν καταστραφεί το αντικείμενο Φοιτητής, το αντικείμενο Καθηγητής θα συνεχίσει να υπάρχει.
Επίσης τυχόν αλλαγές στο αντικείμενο Καθηγητή θα αντικατοπτρίζονται άμεσα σε όλους τους φοιτητές που τον έχουν επιβλέποντα.

```
#include <iostream>
#include <string>
using namespace std;

class Professor
{
public:
	string name;
	string office;

	Professor(string _name, string _office): name(_name), office(_office)
	{
	}

	Professor(): name("N/A"), office("N/A")
	{
	}

	void display()
	{
		cout << "Name: " << name << ", office: " << office << endl;
```

```
	}

};

class Student
{
public:
	string name;
	int code;
	Professor *supervisor;

	Student();
	Student(string _name, int _code, Professor *_supervisor);
	void display();
	void setSupervisor(Professor *p);
};

Student::Student()
{
	name = "N/A"; //not available
	code = 0;
	supervisor = NULL;
}

Student::Student(string _name, int _code, Professor *_supervisor)
{
	name = _name;
	code = _code;
	supervisor = _supervisor;
}

void Student::display()
{
	cout << "Name: " << name << ", code: " << code << endl;
	cout << "Supervisor: " << endl;

	if(supervisor != NULL)
		supervisor->display();
	else
		cout << "No supervisor" << endl;
}

void Student::setSupervisor(Professor *p)
{
	supervisor = p;
}

int main()
{
	Professor p("Henry Jones", "301A");
	Student s("John Doe", 2323, &p);

	s.display();
	cout << "---------------------------------" << endl;
	p.office = "113B"; //change of office
	s.display(); // new office is shown

	cout << "---------------------------------" << endl;
	Professor p1("Jack Welch", "222");
	s.setSupervisor(&p1); //change of supervisor
	s.display(); // new supervisor is shown

	cout << "---------------------------------" << endl;
```

```
	Student s1;
	s1.display();
	system("pause");
	return 0;
}
```

Άσκηση 1: Composition

Θέλουμε στην κλάση Patient να καταγράφουμε μετρήσεις θερμοκρασίας του ασθενούς. Οι μετρήσεις θα είναι πολλές και θα πρέπει να καταγράφεται η ημερομηνία και ώρα λήψης τους.

Για τον σκοπό αυτό θα πρέπει:

- Να υλοποιήσετε μια νέα κλάση Measurement όπου θα καταγράφονται τα εξής:
  - Η θερμοκρασία του ασθενούς
  - Η ημερομηνία και η ώρα λήψης της μέτρησης (για απλότητα χρησιμοποιήστε strings, π.χ. "12/03/2015 23:30")

  Μην ξεχάσετε να υλοποιήσετε και τους απαραίτητους constructors.
- Να προσθέσετε ένα νέο πεδίο στην κλάση Patient, **που θα είναι Vector** και θα περιέχει όλες τις μετρήσεις του ασθενούς
- Να υλοποιήσετε μια συνάρτηση για την εισαγωγή μιας μέτρησης σε αυτό τον πίνακα
  `π.χ. void insertMeasurement(Measurement m);`
- Τέλος, να υλοποιήσετε και μια συνάρτηση maxTemp( ) που θα επιστρέφει την μέγιστη θερμοκρασία που έχει καταγραφεί.

Άσκηση 2: Aggregation

Θέλουμε να επεκτείνουμε την προηγούμενη άσκηση ώστε καταγράψουμε και τον θεράποντα ιατρό του κάθε ασθενούς.
Κάθε ασθενής έχει μόνο ένα ιατρό, ενώ προφανώς κάθε ιατρός παρακολουθεί πολλούς ασθενείς ταυτόχρονα. Για τον σκοπό αυτό θα χρησιμοποιήσουμε συνάθροιση (aggregation).

Θα πρέπει λοιπόν:

- Να ορίσετε μια νέα κλάση Doctor που θα καταγράφει τις βασικές πληροφορίες ενός ιατρού (όνομα, ειδικότητα, τηλέφωνο)
- Να προσθέσετε μια ακόμα μεταβλητή στην κλάση Patient ώστε να αποθηκεύσουμε εκεί τον θεράποντα ιατρό.
- Στην συνάρτηση main, να φτιάξετε ένα ιατρό και 2 ασθενείς και να ορίσετε τον ιατρό αυτό ως θεράποντα και στους δύο ασθενείς

Ερώτηση: Ο γιατρός ξέρει ποιοι είναι οι ασθενείς του; ….

Παράρτημα: Vectors

Οι Vectors θα παρουσιαστούν εκτενώς σε επόμενο μάθημα. Όμως, επειδή αποτελούν πολύ χρήσιμο εργαλείο για την αποθήκευση δεδομένων, καλό είναι να εξοικειωνόμαστε με την χρήση τους.

Στο παρακάτω παράδειγμα αρχικά δημιουργούμε ένα Vector ο οποίος θα περιέχει integers και στην συνέχεια εισάγουμε τιμές σε αυτόν. Γενικά η δήλωση

```
vector<MyClass> v;
```

δημιουργεί έναν Vector με όνομα v που έχει την δυνατότητα να περιέχει αντικείμενα της κλάσης MyClass.

```
#include <iostream>
#include <vector>
using namespace std;

int main()
{
    vector<int> v;            //create a vector of integers

    v.push_back(10);          //put values at end of array
    v.push_back(11);
    v.push_back(12);
    v.push_back(13);

    v[0] = 20;                //replace with new values
    v[3] = 23;

    for(int j=0; j<v.size(); j++) //display vector contents
      cout << v[j] << ' ';        //20 11 12 23
   cout << endl;
   return 0;
}
```

Ουσιαστικά μοιάζει με ένα πίνακα στον οποίο όμως δεν χρειάζεται να ελέγξουμε τα όρια του.

Ένα άλλο πλεονέκτημα είναι η εισαγωγή δεδομένων σε οποιοδήποτε μέρος του Vector:

```
#include <iostream>
#include <vector>
using namespace std;

int main()
{
   int arr[] = { 100, 110, 120, 130 };  //an array of integers

   vector<int> v(arr, arr+4);       //initialize vector to array

   cout << "\nBefore insertion: ";
   for(int j=0; j<v.size(); j++)         //display all elements
      cout << v[j] << ' ';

   v.insert( v.begin()+2, 115);          //insert 115 at element 2

   cout << "\nAfter insertion:  ";
   for(j=0; j<v.size(); j++)             //display all elements
       cout << v[j] << ' ';

   v.erase( v.begin()+2 );               //erase element 2

   cout << "\nAfter erasure:    ";
   for(j=0; j<v.size(); j++)             //display all elements
       cout << v[j] << ' ';
   cout << endl;
   return 0;
  }
```

Για περισσότερες πληροφορίες μπορείτε να ανατρέξετε στο `http://www.cplusplus.com/reference/stl/vector/` όπου θα βρείτε όλες τις συναρτήσεις που παρέχει η κλάση αυτή.

Παρακάτω φαίνονται οι συναρτήσεις τις οποίες υποστηρίζει ένας Vector:

Vector constructors	create vectors and initialize them with some data
Vector operators	compare, assign, and access elements of a vector
assign	assign elements to a vector
at	returns an element at a specific location
back	returns a reference to last element of a vector
begin	returns an iterator to the beginning of the vector
capacity	returns the number of elements that the vector can hold
clear	removes all elements from the vector
empty	true if the vector has no elements
end	returns an iterator just past the last element of a vector
erase	removes elements from a vector
front	returns a reference to the first element of a vector
insert	inserts elements into the vector
max_size	returns the maximum number of elements that the vector can hold
pop_back	removes the last element of a vector
push_back	add an element to the end of the vector
rbegin	returns a reverse_iterator to the end of the vector
rend	returns a reverse_iterator to the beginning of the vector
reserve	sets the minimum capacity of the vector
resize	change the size of the vector
size	returns the number of items in the vector
swap	swap the contents of this vector with another

(Από το site: http://www.cppreference.com/cppvector/index.html)

Άλλα παραδείγματα:

Εισαγωγή αριθμών στο τέλος του vector

```
vector<int> the_vector;
for( int i = 0; i < 10; i++ )
{
  the_vector.push_back( i );
}

for(i = 0; i < 10; i++ )
{
  cout << the_vector.pop_back();
}
```

Αποθήκευση strings στον vector και ταξινόμηση

```
vector<string> words;
string str;

while( cin >> str )
  words.push_back(str);

sort( words.begin(), words.end() );
```

```
  cout << "In alphabetical order, the first word is '" << words.front() << "'." <<
endl;
```

Αποθήκευση **αντικειμένων τύπου Vehicle** στον vector

```
#include <iostream>
#include <string>
#include <vector>

using namespace std;

class Vehicle
{
public:
	string model;

};

int main()
{
	Vehicle v;
	vector<Vehicle> vec;
	v.model = "Yaris";

	vec.push_back(v);

	v.model = "316";
	vec.push_back(v);

	for(int i=0; i<vec.size(); i++)
		cout << "Model: " << vec[i].model << endl;

	system("pause");

	return 0;
}
```