# Εργαστήριο 11ο

*Inheritance*

## Άσκηση 1 (inheritance basics)

Να δημιουργήσετε το παρακάτω πρόγραμμα Lab10E1 και σε αυτό να προσθέσετε μία κλάση Circle που θα περιέχει

Μεταβλητές μέλη

- double radius

Μεθόδους

- void set_radius(double d) –θέτει τιμή στην μεταβλητή μέλος
- double get_radius() const – επιστρέφει την τιμή της μεταβλητής μέλους
- double get_area() const – επιστρέφει την επιφάνεια του κύκλου
- double get_perimeter() const – επιστρέφει την περίμετρο του κύκλου
- υπερφόρτωση του τελεστή εξόδου όπου θα τυπώνει ότι είναι κύκλος και την τιμή της ακτίνας

Να δοκιμάσετε την κλάση που δημιουργήσατε με την παρακάτω main

```
int main()
{
	Circle c1;
	c1.set_radius(5);

	cout << c1;
	cout << "Epifaneia "<<c1.get_area()<<endl;
	cout << "Perimetros " << c1.get_perimeter() << endl;
}
```

Στην συνέχεια θα δημιουργήσουμε μια κλάση Cylinder που παράγεται από την κλάση Circle. Ο κώδικας της είναι ο παρακάτω:

```
----- Αρχείο Cylinder.h
#include "Circle.h"
#include<iostream>
using namespace std;
class Cylinder :
	public Circle
{
public:
	void set_height(double h);
	double get_height() const;
	double calc_volume();
	friend ostream &operator<<(ostream &out, const Cylinder &it);
private:
	double height;
};

------ Αρχείο Cylinder.cpp
#include "pch.h"
#include "Cylinder.h"

void Cylinder::set_height(double h)
```

```
{
	height = h;
}

double Cylinder::get_height() const
{
	return height;
}

double Cylinder::calc_volume()
{
	return 3.14*this->get_radius()*this->get_radius()*height;
}

ostream & operator<<(ostream & out, const Cylinder & it)
{
	out << "I am a Cylinder\n";
	out << "Radius:" << it.get_radius() << " Height:" << it.get_height()<<endl;
	return out;
}

---- Αρχείο main
int main()
{
	Circle c1;
	c1.set_radius(5);
	Cylinder c2;
	c2.set_radius(5);
	c2.set_height(6);

	cout << c1;
	cout << "Epifaneia "<<c1.get_area()<<endl;
	cout << "Perimetros " << c1.get_perimeter() << endl;

	cout << c2;
	cout << "Epifaneia " << c2.get_area() << endl;
	cout << "Perimetros " << c2.get_perimeter() << endl;
	cout << "Ogkos " << c2.calc_volume() << endl;
}
```

Να απαντήσετε στις παρακάτω ερωτήσεις:

1. Ποιες είναι οι δύο κλάσεις που δημιουργούμε στο πρόγραμμα μας; Ποια είναι τα objects των κλάσεων αυτών;
2. Μπορούμε να πούμε ότι έχουν κάποια σχέση το c2 και το c1;
3. Δηλώνουμε λοιπόν μια κλάση (derived, child) που βασίζεται σε μία άλλη κλάση (base, parent) και περιέχει και μερικά επιπρόσθετα στοιχεία. Στο παράδειγμα μας ποια είναι η base και ποια είναι η derived class;
4. Με ποιον τρόπο ορίζουμε στην C++ ότι μια class B γίνεται derived από μία άλλη A;
5. Πόσα επιπλέον member variables και methods έχει η class cylinder;

Παρατηρούμε ότι η calc_volume αν και είναι μέθοδος για να έχει πρόσβαση στην μεταβλητή-μέλος radius χρησιμοποιεί την μέθοδο get_radius. Να πραγματοποιήσετε τις παρακάτω αλλαγές

1. Τροποποιείστε τον ορισμό της calc_volume όπως παρακάτω:

```
double Cylinder::calc_volume()
{
	return PI * radius * radius * height;
}
```

1. Μπορεί να εκτελεστεί το πρόγραμμα; Τι συμπέρασμα βγάζετε σχετικά με την πρόσβαση στην private member variable radius;
2. Δοκιμάστε στο class Circle το private να το κάνετε protected. Μπορεί τώρα να εκτελεστεί το πρόγραμμα;

| **ΒΑΣΙΚΕΣ ΕΝΝΟΙΕΣ ΤΗΣ ΑΣΚΗΣΗΣ** |
|---|
| **Πως δηλώνω στην C++ ότι μια class B γίνεται derived από μία class A;** |
| |
| **Ποια member variables έχει η παραπάνω class B;** |
| |
| **Σε ποια στοιχεία του A έχει πρόσβαση η B;** |
| |
| **Ποια είναι η λειτουργία του protected qualifier;** |
| |

## Άσκηση 2 (constructors - destructors)

Εισάγετε τους παρακάτω default constructors και τους destructors:

```
Circle::Circle(): radius(0)
{
	cout << "Constructor Circle Default\n";
}

Circle::~Circle()
{
	cout << "Destroying Circle\n";
}

Cylinder::Cylinder() : height(0)
{
	cout << "Constructor Cylinder Default\n";
}

Cylinder::~Cylinder()
{
	cout << "Destroying Cylinder\n";
}
```

Να απαντήσετε στις παρακάτω ερωτήσεις;

1. Με ποια σειρά καλούνται οι constructors για το object cyl1;

2. Με ποια σειρά καλούνται οι destructors για το object cyl1;
3. Σαν γενικό συμπέρασμα βγάζουμε ότι
    a. όταν σε έναν constructor μιας derived κλάσης δεν καλούμε τον base τότε καλείται αυτόματα ο …….
    b. Οι constructors και destructors της base class …… κληρονομούνται στην derived

Εισάγετε και τους παρακάτω non default constructors

```
Circle::Circle(double r) : radius(r)
{
}

Cylinder::Cylinder(double r, double h): Circle(r), height(h)
{
}
```

Να απαντήσετε στις παρακάτω ερωτήσεις:

1. Με ποιον τρόπο καλούμε τον constructor της base class μέσα από τον constructor της derived;

Να πραγματοποιήσετε τις παρακάτω αλλαγές

1. Τροποποιείστε τον κώδικα της main έτσι ώστε να δημιουργούνται τα objects c1 και cyl1 με τις ίδιες τιμές στις member variables χωρίς όμως την χρήση των methods set.

## Άσκηση 3 (Redefinition of methods)

Μέχρι τώρα στην άσκηση κάναμε μια μικρή παρατυπία. Όταν ένα Cylinder object καλούσε την calc_area τότε αυτή εκτελούσε τον κώδικα που ήταν ορισμένος μέσα στην Circle class και στην ουσία επέστρεφε το εμβαδό του κύκλου της μίας βάσης.
Προκειμένου να διορθώσουμε αυτό το λάθος προσθέτουμε στην Cylinder class την μέθοδο calc_area με τον παρακάτω κώδικα.

```
double Cylinder::get_area()
{
	cout << "Area for cylinder\n";
	return (this->get_perimeter()*(radius*height));
}
```

Απαντήστε στις ερωτήσεις:

a) Μπορούμε να έχουμε μια method με το ίδιο όνομα τόσο στην Base όσο και στην derived class;
b) Αν ναι, πως καταλαβαίνει η C++ ποια method θα καλέσει;

| ΠΡΟΣΟΧΗ! Δεν είναι η πρώτη φορά που βλέπουμε methods με το ίδιο όνομα, το είχαμε δει και στην τεχνική του overloading. Μπορείτε να αναφέρετε τις βασικές διαφορές ανάμεσα στο method overloading και method redefinition; |
| --- |
| |

Μπορούμε τον παραπάνω κώδικα της calc_area για την Cylinder class να τον κάνουμε ακόμα πιο έξυπνο. Αν σκεφτούμε καλό η επιφάνεια ενός κώνου ισούται με δύο φορές την επιφάνεια της βάσης συν την πλευρική επιφάνεια. Επομένως, μπορούμε στην calc_area της Cylinder class να

χρησιμοποιήσουμε την calc_area της Circle μιας και αυτή περιγράφει την επιφάνεια της βάσης. Έχουμε όμως δύο προβλήματα

a) πως θα καλέσουμε μια μέθοδο μέσα από μία μέθοδο; Που βρίσκεται το object που χρειαζόμαστε για να καλέσουμε την μέθοδο;

   Η λύση βρίσκεται στο γεγονός ότι κάθε μέθοδος που καλείται έχει μια «κρυμμένη» παράμετρο με το object το οποίο την κάλεσε. Αυτή η παράμετρος γίνεται με κλήση με αναφορά (δηλαδή είναι pointer) και έχει το όνομα this. Επομένως, με this->method μπορούμε να καλέσουμε όποια μέθοδο θέλουμε από το object που την κάλεσε.

b) Πάλι όμως έχουμε το πρόβλημα ότι ο κώδικας this->calc_area() καλεί την μέθοδο calc_area της κλάσης Cylinder (δηλαδή τον εαυτό της). Πως μπορούμε να καλέσουμε την μέθοδο της base class Circle;

Η λύση βρίσκεται στην χρήση του scope operator (::). Αν κάνουμε κλήση object.method(), τότε καλείται η μέθοδος της κλάσης στην οποία ανήκει το αντικείμενο (Cylinder). Με την χρήση όμως του :: μπορούμε να καλέσουμε την μέθοδο του Base class (Circle). Επομένως η κλήση μπορεί να γίνει this->Circle::calc_area()

## Άσκηση 4 (Late binding)

Ας υποθέσουμε ότι έχουμε μια συνάρτηση paint που παίρνει σαν παράμετρο ένα object της κλάσης Circle και μέσα σε αυτή καλείται η calc_area method. Προσθέστε την συνάρτηση με τον παρακάτω ορισμό (την δήλωση συμπληρώστε την μόνοι σας)

```
void paint(Circle x)
{
	cout << "I will paint " << x.get_area() << " square meters\n";
}
```

Επίσης αντικαταστήστε την main με τον κώδικα που περιγράφεται παρακάτω:

```
int main()
{
	Circle c1(1);
	cout << "Circle c1 has:\n";
	cout << "area: " << c1.calc_area() << endl << endl;

	Cylinder cyl1(1,2);
	cout << "Cylinder cyl1 has:\n";
	cout << "area: " << cyl1.calc_area() << endl;

	cout << "calling paint\n";
	paint(c1);
	paint(cyl1);
	return 0;
}
```

Απαντήστε στις ερωτήσεις:

a) Γιατί λειτουργεί η εντολή paint(cyl1); Αφού η συνάρτηση paint λαμβάνει σαν ορίσματα μόνο object της Circle class και εμείς της δίνουμε όρισμα Cylinder class;

b) Έχει διαφορά το αποτέλεσμα κλήση της c1 και της cyl1;

Επομένως συμπεραίνουμε ότι ένα Derived class object είναι στην ουσία και ένα ..... και μπορούμε να το τοποθετήσουμε στην θέση του. Δεν ισχύει όμως το αντίθετο!

Πραγματοποιείστε τις παρακάτω αλλαγές:

1) Προσθέστε την λέξη virtual πριν από την δήλωση της calc_area method στην Circle class. Τι παρατηρείτε;

## Άσκηση 5

Να δημιουργήσετε μια εφαρμογή για ένα πανεπιστήμιο. Αυτό έχει Students που χωρίζονται σε Undergraduate και Graduate. Για κάθε φοιτητή θα υπάρχει id (int) και name (string). Τώρα ο κάθε Undergraduate θα έχει ένα vector grades που θα έχει τους βαθμούς σε ακεραίους από το 1 μέχρι το 10. Στους undergraduate στο vector οι βαθμοί θα αποθηκεύονται με A-D (chars). Η αντιστοίχιση θα γίνεται ως εξής A(10-9) B(8-7) C(6-5) D (όλα τα υπόλοιπα). Να δημιουργήσετε:

1) Μία μέθοδο set_grade που θα παίρνει όρισμα ακέραιο και θα το αποθηκεύει κατάλληλα σε κάθε φοιτητή
2) Μία μέθοδο get_av που θα επιστρέφει το μέσο όρο του φοιτητή (προσοχή και οι Postgraduate θα επιστρέφουν double, η αντιστοίχιση είναι A=10, B=8, C=6, D=4)
3) Να υπερφορτώσετε τον τελεστή < που ελέγχει δύο φοιτητές όσο αφορά τον μέσο όρο.