**Ανοικτά Ακαδημαϊκά Μαθήματα**

**Τεχνολογικό Εκπαιδευτικό Ίδρυμα Αθήνας**

**Αντικειμενοστρεφής Προγραμματισμός (Ε)**

**Ενότητα 3: κληρονομικότητα**

Κλειώ Σγουροπούλου

Τμήμα Μηχανικών Πληροφορικής ΤΕ

|  |  |
| --- | --- |
| Το περιεχόμενο του μαθήματος διατίθεται με άδεια Creative Commons εκτός και αν αναφέρεται διαφορετικά | Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «Εκπαίδευση και Δια Βίου Μάθηση» και συγχρηματοδοτείται από την Ευρωπαϊκή Ένωση (Ευρωπαϊκό Κοινωνικό Ταμείο) και από εθνικούς πόρους. |

Πίνακας Περιεχομένων

[Αντικειμενοστρεφής Προγραμματισμός- Εργαστήριο 3 3](#_Toc402874397)

[Κληρονομικότητα 3](#_Toc402874398)

[**Παράδειγμα 1** 4](#_Toc402874399)

[**Άσκηση 1** 4](#_Toc402874400)

[**Παράδειγμα 2** 5](#_Toc402874401)

[**Παράδειγμα 3** 5](#_Toc402874402)

[**Παράδειγμα 4** 6](#_Toc402874403)

[**Άσκηση 2** 7](#_Toc402874404)

[**Παράδειγμα 5** 8](#_Toc402874405)

[**Άσκηση 3** 10](#_Toc402874406)

[Πολλαπλή Κληρονομικότητα 10](#_Toc402874407)

[Λίγα λόγια για την UML 11](#_Toc402874408)

[Inheritance, Composition και Aggregation (Κληρονομικότητα, Σύνθεση, Συνάρθοιση) 11](#_Toc402874409)

# Αντικειμενοστρεφής Προγραμματισμός- Εργαστήριο 3

## Κληρονομικότητα

Η κληρονομικότητα αποτελεί ένα από τα κυριότερα χαρακτηριστικά μιας αντικειμενοστρεφούς γλώσσας προγραμματισμού. Στη C++ η κληρονομικότητα επιτυγχάνεται με το να επιτρέπεται σε μια κλάση να κληρονομεί και να διαχειρίζεται τα στοιχεία μιας άλλης κλάσης μέσα από τη διακήρυξη της. Η πρώτη κλάση ονομάζεται παραγόμενη (derived class) και η δεύτερη βασική (base class).

Η γενική μορφή της διακήρυξης μιας παραγόμενης κλάσης είναι:

class όνομα παραγόμενης κλάσης **: τύπος προσπέλασης** όνομα βασικής κλάσης{…}

O τύπος προσπέλασης δηλώνει τον τρόπο προσπέλασης των στοιχείων της βασικής κλάσης από την παραγόμενη και είναι ένας από τις λέξεις κλειδιά: public, protected ή private.

Στη C++ οι κλάσεις αναγνωρίζουν τρεις κατηγορίες στοιχείων : public, protected και private. Τα public στοιχεία μιας κλάσης είναι προσπελάσιμα από κάθε συνάρτηση του προγράμματος. Τα private στοιχεία είναι προσπελάσιμα μόνο από τις συναρτήσεις μέλη ή από τις φιλικές συναρτήσεις της κλάσης. Τα στοιχεία που διακηρύσσονται ως protected είναι προσπελάσιμα μόνο από τις συναρτήσεις μέλη και τις φιλικές συναρτήσεις της κλάσης καθώς και *από τις από τις αντίστοιχες συναρτήσεις των παραγόμενων από αυτή κλάσεων*.

Εάν κατά τη διακήρυξη μιας παραγόμενης κλάσης από μια βασική ο τύπος προσπέλασης είναι:

1. Public:

Τότε όλα τα public και protected στοιχεία της βασικής κλάσης γίνονται αντίστοιχα public και protected στοιχεία της παραγόμενης κλάσης.

1. Protected:

Τότε όλα τα public και protected στοιχεία της βασικής κλάσης γίνονται protected στοιχεία της παραγόμενης κλάσης.

1. Private:

Τότε όλα τα public και protected στοιχεία της βασικής κλάσης γίνονται private στοιχεία της παραγόμενης κλάσης.

Σημειώνεται ότι δεν υπάρχει η δυνατότητα προσπέλασης στα private στοιχεία της βασικής κλάσης.

Ο παρακάτω πίνακας συνοψίζει τις 3x3 περιπτώσεις:

|  |  |  |  |
| --- | --- | --- | --- |
| **Access** | **public** | **protected** | **private** |
| Μέλη της ίδιας κλάσης | Ναι | Ναι | Ναι |
| Μέλη παραγόμενης κλάσης | Ναι | Ναι | Όχι |
| Μη μέλη | Ναι | Όχι | Όχι |

**Παραδείγματα:**

### **Παράδειγμα 1**

Σε αυτό το παράδειγμα, οι δύο μεταβλητές της Shape είναι public, συνεπώς είναι ορατές και από τα μέλη της

Triangle και από τον «έξω κόσμο», δηλ. την main.

#include <iostream>

#include <string>

using namespace std;

// A class for two-dimensional objects. class Shape {

**public:**

double width; double height;

void showDim() {

cout << "Width and height are " <<

width << " and " << height << "\n";

}

};

// Triangle is derived from Shape. class Triangle : **public** Shape { public:

string style;

double area() {

return width \* height / 2;

}

void showStyle() {

cout << "Triangle is " << style << "\n";

}

};

int main() { Triangle t1;

t1.width = 4.0;

t1.height = 4.0;

t1.style = "isosceles";

cout << "Info for t1:\n"; t1.showStyle(); t1.showDim();

cout << "Area is " << t1.area() << "\n";

return 0;

}

### **Άσκηση 1**

Θεωρήστε μια κάβα που πουλά κρασί και οινοπνευματώδη ποτά. Προσπαθήστε να υλοποιήσετε την

απαραίτητη ιεραρχία κλάσεων ώστε οι κοινές πληροφορίες (π.χ. παραγωγός, τιμή λιανικής πώλησης) να μην επαναλαμβάνονται στις παραγόμενες κλάσεις.

Σημειώστε ότι όλα τα προιόντα προς πώληση έχουν τα εξής κοινά χαρακτηριστικά:

* Όνομα του κρασιού/ποτού
* Όνομα παραγωγού
* Τιμή πώλησης

Επιπλέον, τα κρασιά θα πρέπει να περιέχουν πληροφορία για την ποικιλία του σταφυλιού από την οποία προέρχονται.

Επίσης, τα ποτά θα πρέπει να περιέχουν πληροφορία για το είδος τους (ουίσκι, βότκα κλπ).

### **Παράδειγμα 2**

Σε αυτό το παράδειγμα, οι δύο μεταβλητές της Shape είναι private (by default), συνεπώς δεν είναι ορατές ούτε

από τα μέλη της Triangle ούτε βεβαίως από τον «έξω κόσμο».

// Access to private members is not granted to derived classes. class Shape {

// these are now private

double width; double height;

public:

void showDim() {

cout << "Width and height are " <<

width << " and " << height << "\n";

}

};

// Triangle is derived from Shape. class Triangle : public Shape { public:

string style;

double area() {

return width \* height / 2;

}

void showStyle() {

cout << "Triangle is " << style << "\n";

}

};

### **Παράδειγμα 3**

Μια λύση για να αποκτήσουμε πρόσβαση στις δύο μεταβλητές της Shape είναι να χρησιμοποιήσουμε accessor

functions (που θα είναι βεβαίως public).

#include <iostream>

#include <string>

using namespace std;

// A class for two-dimensional objects. class Shape {

// these are private

double width;

double height;

public:

void showDim() {

cout << "Width and height are " <<

width << " and " << height << "\n";

}

// accessor functions

double getWidth() { return width; }

double getHeight() { return height; }

void setWidth(double w) { width = w; }

void setHeight(double h) { height = h; }

};

// Triangle is derived from Shape. class Triangle : public Shape { public:

string style;

double area() {

return getWidth() \* getHeight() / 2;

}

void showStyle() {

cout << "Triangle is " << style << "\n";

}

};

int main() { Triangle t1;

t1.setWidth(4.0); t1.setHeight(4.0); strcpy(t1.style, "isosceles");

cout << "Info for t1:\n"; t1.showStyle(); t1.showDim();

cout << "Area is " << t1.area() << "\n";

return 0;

}

### **Παράδειγμα 4**

Η άλλη λύση για να αποκτήσουμε πρόσβαση στις δύο μεταβλητές της Shape είναι να τις δηλώσουμε ως

protected που είναι μια ενδιάμεση κατάσταση μεταξύ public και private. Πιο συγκεκριμένα, μια τέτοια μεταβλητή είναι με ορατή από την θυγατρική κλάση αλλά όχι από την main.

#include <iostream>

#include <cstring> using namespace std;

// A class for two-dimensional objects. class Shape {

**protected:**

double width; double height;

public:

// accessor functions

double getWidth() { return width; }

double getHeight() { return height; }

void setWidth(double w) { width = w; }

void setHeight(double h) { height = h; }

void showDim() {

cout << "Width and height are " <<

width << " and " << height << "\n";

}

};

// Triangle is derived from Shape.

class Triangle : **public** Shape { public:

char style[20];

double area() {

return width \* height / 2;

}

void showStyle() {

cout << "Triangle is " << style << "\n";

}

};

int main() { Triangle t1;

t1.setWidth(4.0); t1.setHeight(4.0); strcpy(t1.style, "isosceles");

cout << "Info for t1:\n"; t1.showStyle(); t1.showDim();

cout << "Area is " << t1.area() << "\n";

return 0;

}

### **Άσκηση 2**

Να βρείτε τα λάθη του παρακάτω προγράμματος και να κάνετε τις απαραίτητες διορθώσεις

**#include** <iostream>

**using** namespace std;

**class** A

{

**private**:

**int** var\_a;

**void** show();

**public**:

**void** check\_B();

};

**class** B : **public** A

{

**public**:

**int** var\_b;

**void** display();

};

**void** A::show()

{

cout << var\_a << endl;

}

**void** A::check\_B()

{

cout << var\_b << endl;

}

**void** B::display()

{

cout << var\_a << ' ' << var\_b << ' ' << endl;

}

**int** main()

{

B b;

cin >> b.var\_b; cin >> b.var\_a;

b.show();

b.display();

**return** 0;

}

### **Παράδειγμα 5**

Η αρχικοποίηση των μεταβλητών της Shape και της Triangle μπορεί (και πρέπει) να γίνει με την χρήση

κατασκευαστών. Θεωρητικά, ο κατασκευαστής της Triangle θα μπορούσε να αναλάβει να αρχικοποιήσει όλες τις μεταβλητές (αφού έχει πρόσβαση σε όλες), αλλά το σωστό είναι κάθε κλάση να αρχικοποιεί τις δικές της μεταβλητές με τον δικό της constructor. Αυτό που γίνεται στο τέλος, είναι ότι ο κατασκευαστής του Triangle πρέπει να καλέσει το αντίστοιχο της Shape.

Κάτι που θέλει προσοχή εδώ είναι το ότι ο default constructor της Triangle καλεί αυτόματα τον αντίστοιχο της Shape.

#include <iostream>

#include <cstring>

using namespace std;

class Shape { double width; double height;

public:

// Default constructor. Shape() {

width = height = 0.0;

}

// Constructor for Shape. Shape(double w, double h) {

width = w; height = h;

}

// Construct object with equal width and height. Shape(double x) {

width = height = x;

}

void showDim() {

cout << "Width and height are " <<

width << " and " << height << "\n";

}

// accessor functions

double getWidth() { return width; }

double getHeight() { return height; }

void setWidth(double w) { width = w; } void setHeight(double h) { height = h; }

};

class Triangle : public Shape { char style[20]; // now private

public:

/\* A default constructor. This automatically invokes the default constructor of Shape. \*/

Triangle() {

strcpy(style, "unknown");

}

// Constructor with three parameters.

Triangle(char \*str, double w, double h) : Shape(w, h) {

strcpy(style, str);

}

// Construct an isosceles triangle. Triangle(double x) : Shape(x) {

strcpy(style, "isosceles");

}

double area() {

return getWidth() \* getHeight() / 2;

}

void showStyle() {

cout << "Triangle is " << style << "\n";

}

};

int main() { Triangle t1;

Triangle t2("right", 8.0, 12.0); Triangle t3(4.0);

t1 = t2;

cout << "Info for t1: \n"; t1.showStyle(); t1.showDim();

cout << "Area is " << t1.area() << "\n";

cout << "\n";

cout << "Info for t2: \n"; t2.showStyle(); t2.showDim();

cout << "Area is " << t2.area() << "\n"; cout << "\n";

cout << "Info for t3: \n";

t3.showStyle();

t3.showDim();

cout << "Area is " << t3.area() << "\n"; cout << "\n";

return 0;

}

### **Άσκηση 3**

Συνεχίζοντας την Άσκηση 1, να «κρύψετε» τις μεταβλητές όλων των κλάσεων ώστε να μην είναι προσβάσιμες άμεσα από τον «έξω κόσμο».

Επιπλέον, να υλοποιήσετε τους απαραίτητους κατασκευαστές και των 3 κλάσεων.

Τέλος, στην συνάρτηση main, να φτιάξετε από ένα αντικείμενο κρασιού και ποτού (παίρνοντας input από τον χρήστη) και να εκτυπώσετε στην οθόνη πληροφορίες για κάθε ένα από αυτά, με χρήση κατάλληλης συνάρτησης.

## Πολλαπλή Κληρονομικότητα

Είναι δυνατόν μια κλάση να παράγεται από περισσότερες από μια βασικές κλάσεις. Στην περίπτωση αυτή η νέα κλάση έχει προσπέλαση στα στοιχεία όλων των βασικών κλάσεων σύμφωνα με τους κανόνες της κληρονομικότητας.

Η γενική μορφή της διακήρυξης μιας παραγόμενης κλάσης, με τον ίδιο τύπο προσπέλασης, είναι:

class όνομα παραγόμενης κλάσης : τύπος προσπέλασης όνομα βασικής κλάσης1,…, όνομα βασικής κλάσηςn{…}

Εάν οι βασικές κλάσεις περιέχουν κατασκευαστές που δεν δέχονται παραμέτρους, τότε η παραγόμενη κλάση δεν είναι απαραίτητο να έχει κατασκευαστή. Οι κατασκευαστές των βασικών κλάσεων θα εκτελεστούν αυτόματα με τη δημιουργία του αντικειμένου της παραγόμενης κλάσης. Η προτεραιότητα εκτέλεσης είναι από αριστερά προς τα δεξιά με τη σειρά που εμφανίζονται οι βασικές κλάσεις στη διακήρυξη της παραγόμενης.

Στην περίπτωση που οι κατασκευαστές των βασικών κλάσεων δέχονται παραμέτρους, πρέπει οπωσδήποτε η παραγόμενη κλάση να περιέχει κατασκευαστή.

Παράδειγμα

#include <iostream> using namespace std;

class A{

protected:

int a;

public:

};

A(int m);

class B{

protected:

int b;

public:

};

B(int m);

class C:public A, public B{ private:

int k;

public:

};

C(int i, int j); int set\_k(void);

A::A(int m){

a=m;

cout<<"a="<<a<<"\n";

}

B::B(int m){

b=m; cout<<"b="<<b<<"\n";

}

C::C(int i,int j):A(i),B(j){

cout<<"To a kai to b phran times mesa apo ti C \n";

}

int C::set\_k(void){

k=a+b;

return k;

}

int main(){

int k,i;

k=10; i=20;

C c(k,i); cout<<"k="<<c.set\_k()<<endl;

system("PAUSE"); return 1;

}

Υπάρχει επίσης η περίπτωση οι δύο μητρικές κλάσεις Α και Β να έχουν ως μέλος την ίδια παράμετρο (π.χ. int

1. και αυτή να χρησιμοποιείται από αντικείμενα του τύπου C. Σε αυτή την περίπτωση, μιλάμε για δύο διαφορετικές μεταβλητές που έχουν όμως το ίδιο όνομα. Για να διακρίνουμε μεταξύ τους χρησιμοποιούμε τον

scope operator ::

Δηλαδή: A::i και B::i

## 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 αντικέιμενο 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 αντικέιμενο 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.



|  |
| --- |
|  |
|  |  |
|  |  |
|  |
|  |

|  |
| --- |
| **Professor** |
|  |
|  |

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.

|  |  |
| --- | --- |
|  | **Class** |
|  |
| 1 \* |
|  |

|  |
| --- |
| **House** |
|  |
|  |

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

|  |  |
| --- | --- |
|  | **Room** |
|  |
| 1 \* |
|  |

class House

{

private:

Room rooms[10];

};

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

Ένας από τους τρόπους αντιμετώπισης της συνάθροισης είναι ο ακόλουθος:

class Professor

{

private:

Lecture\* lectures[10];

public:

void addLecture(Lecture \*lec)

{

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 σωστές λογικά. 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).

|  |
| --- |
| **Ανοικτά Ακαδημαϊκά Μαθήματα****Τεχνολογικό Εκπαιδευτικό Ίδρυμα Αθήνας** |
| **Τέλος Ενότητας** |
| **Χρηματοδότηση*** Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στα πλαίσια του εκπαιδευτικού έργου του διδάσκοντα.
* Το έργο «**Ανοικτά Ακαδημαϊκά Μαθήματα στο ΤΕΙ Αθήνας**» έχει χρηματοδοτήσει μόνο τη αναδιαμόρφωση του εκπαιδευτικού υλικού.
* Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «Εκπαίδευση και Δια Βίου Μάθηση» και συγχρηματοδοτείται από την Ευρωπαϊκή Ένωση (Ευρωπαϊκό Κοινωνικό Ταμείο) και από εθνικούς πόρους.

 |

**Σημειώματα**

**Σημείωμα Αναφοράς**

Copyright ΤΕΙ Αθήνας, Κλειώ Σγουροπούλου, 2014. Κλειώ Σγουροπούλου. «Αντικειμενοστρεφής Προγραμματισμός». Ενότητα 3: «Κληρονομικότητα». Έκδοση: 1.0. Αθήνα 2014. Διαθέσιμο από τη δικτυακή διεύθυνση: [ocp.teiath.gr](https://ocp.teiath.gr/).

**Σημείωμα Αδειοδότησης**

Το παρόν υλικό διατίθεται με τους όρους της άδειας χρήσης Creative Commons Αναφορά, Μη Εμπορική Χρήση Παρόμοια Διανομή 4.0 [1] ή μεταγενέστερη, Διεθνής Έκδοση. Εξαιρούνται τα αυτοτελή έργα τρίτων π.χ. φωτογραφίες, διαγράμματα κ.λ.π., τα οποία εμπεριέχονται σε αυτό και τα οποία αναφέρονται μαζί με τους όρους χρήσης τους στο «Σημείωμα Χρήσης Έργων Τρίτων».



[1] http://creativecommons.org/licenses/by-nc-sa/4.0/

Ως **Μη Εμπορική** ορίζεται η χρήση:

* που δεν περιλαμβάνει άμεσο ή έμμεσο οικονομικό όφελος από την χρήση του έργου, για το διανομέα του έργου και αδειοδόχο
* που δεν περιλαμβάνει οικονομική συναλλαγή ως προϋπόθεση για τη χρήση ή πρόσβαση στο έργο
* που δεν προσπορίζει στο διανομέα του έργου και αδειοδόχο έμμεσο οικονομικό όφελος (π.χ. διαφημίσεις) από την προβολή του έργου σε διαδικτυακό τόπο

Ο δικαιούχος μπορεί να παρέχει στον αδειοδόχο ξεχωριστή άδεια να χρησιμοποιεί το έργο για εμπορική χρήση, εφόσον αυτό του ζητηθεί.

**Διατήρηση Σημειωμάτων**

* Οποιαδήποτε αναπαραγωγή ή διασκευή του υλικού θα πρέπει να συμπεριλαμβάνει:
* Το Σημείωμα Αναφοράς
* Το Σημείωμα Αδειοδότησης
* Τη δήλωση Διατήρησης Σημειωμάτων
* Το Σημείωμα Χρήσης Έργων Τρίτων (εφόσον υπάρχει) μαζί με τους συνοδευόμενους υπερσυνδέσμους.