Αντικειμενοστραφείς Γλώσσες Προγραμματισμού C++ / ROOT

Ιωάννης Παπαδόπουλος

Τμήμα Φυσικής, Πανεπιστήμιο Ιωαννίνων

Δεκέμβριος 2024

Περιεχόμενα
Σύνθεση
Κληρονομικότητα
Υπερφόρτωση
Πολυμορφισμός
Αφηρημένες κλάσεις
Σημαιες κλάσεις

Περιεχόμενα

Περιεχόμενα

Περιεχόμενα
Σύνθεση
Κληρονομικότητα
Υπερφόρτωση
Πολυμορφισμός
Αφηρημένες κλάσεις
Συμπαγείς κλάσεις

1. Σύνθεση (composition)
2. Κληρονομικότητα (inheritance)
   - Υπερφόρτωση κληρονομημένων μελών
3. Πολυμορφισμός (polymorphism)
   - Αφηρημένες κλάσεις (abstract classes)
   - Συμπαγείς κλάσεις (concrete classes)

Σύνθεση (composition)

Σύνθεση (composition)

- Επιτρέπει τη χρήση μιας ή περισσοτέρων κλάσεων στον ορισμό μιας νέας κλάσης.

```
1  //--- file: 08-Point2D.h
2  #pragma once
3
4  #include <iostream>
5
6  class Point2D {
7    private:
8      double x;
9      double y;
10   public:
11     Point2D();
12     Point2D(const double& x, const double& y);
13     ~Point2D();
14     double& getx();
15     double& gety();
16     void setx(const double& x);
17     void sety(const double& y);
18     friend std::ostream& operator<< (std::ostream& o, const Point2D& p);
19 };
```

Σύνθεση (composition)

- Επιτρέπει τη χρήση μιας ή περισσοτέρων κλάσεων στον ορισμό μιας νέας κλάσης.
- Το αντικείμενο μίας κλάσης είναι σύνθετο (composite) όταν στα δεδομένα του περιλαμβάνονται αντικείμενα άλλων κλάσεων.

```
1  //--- file: 08-Point2D.h
2  #pragma once
3
4  #include <iostream>
5
6  class Point2D {
7    private:
8      double x;
9      double y;
10   public:
11     Point2D();
12     Point2D(const double& x, const double& y);
13     ~Point2D();
14     double& getx();
15     double& gety();
16     void setx(const double& x);
17     void sety(const double& y);
18     friend std::ostream& operator<< (std::ostream& o, const Point2D& p);
19 };
```

Σύνθεση (composition)

- Επιτρέπει τη χρήση μιας ή περισσοτέρων κλάσεων στον ορισμό μιας νέας κλάσης.
- Το αντικείμενο μίας κλάσης είναι σύνθετο (composite) όταν στα δεδομένα του περιλαμβάνονται αντικείμενα άλλων κλάσεων.
- παράδειγμα: χρήση της κλάσης Point2D εντός της σύνθετης κλάσης Circle

```
1  //--- file: 08-Point2D.h
2  #pragma once
3
4  #include <iostream>
5
6  class Point2D {
7    private:
8      double x;
9      double y;
10   public:
11     Point2D();
12     Point2D(const double& x, const double& y);
13     ~Point2D();
14     double& getx();
15     double& gety();
16     void setx(const double& x);
17     void sety(const double& y);
18     friend std::ostream& operator<< (std::ostream& o, const Point2D& p);
19 };
```

Σύνθεση (composition)

- Επιτρέπει τη χρήση μιας ή περισσοτέρων κλάσεων στον ορισμό μιας νέας κλάσης.
- Το αντικείμενο μίας κλάσης είναι σύνθετο (composite)
  όταν στα δεδομένα του περιλαμβάνονται αντικείμενα άλλων κλάσεων.
- παράδειγμα: χρήση της κλάσης Point2D εντός της σύνθετης κλάσης Circle

```
1  //--- file: 08-Circle.h
2  #pragma once
3
4  #include "08-Point2D.h"
5
6  class Circle {
7    private:
8      double R;
9      Point2D C;
10   public:
11     Circle();
12     Circle(const double&  R, const Point2D& C);
13     Circle(const Point2D& C, const double&  R);
14     Circle(const double& R, const double& x, const double& y);
15     ~Circle();
16     double& getR();
17     Point2D& getC();
18     void setR(const double&  R);
19     void setC(const Point2D& C);
20     friend std::ostream& operator<< (std::ostream& o, const Circle& p);
21 };
```

Σύνθεση (composition)

```cpp
//--- file: 08-Point2D.cpp
#include "08-Point2D.h"

Point2D::Point2D() : x(0), y(0) {}
Point2D::Point2D(const double& x, const double& y) : x(x), y(y) {}
Point2D::~Point2D() {}
double& Point2D::getx() {
  return x;
}
double& Point2D::gety() {
  return y;
}
void Point2D::setx(const double& x) {
  this->x = x;
}
void Point2D::sety(const double& y) {
  this->y = y;
}
std::ostream& operator<<(std::ostream& o, const Point2D& p) {
  o << "(" << p.x << ", " << p.y << ")";
  return o;
}
```

Σύνθεση (composition)

```
1  //--- file: 08-Circle.cpp
2  #include "08-Circle.h"
3
4  Circle::Circle() : R(0), C(Point2D()) {}
5  Circle::Circle(const double& R, const Point2D& C) : R(R), C(C) {}
6  Circle::Circle(const Point2D& C, const double& R) : R(R), C(C) {}
7  Circle::Circle(const double& R, const double& x, const double& y)
8                 : R(R), C(Point2D(x, y)) {}
9  Circle::~Circle() {}
10 double& Circle::getR() {
11   return R;
12 }
13 Point2D& Circle::getC() {
14   return C;
15 }
16 void Circle::setR(const double& R) {
17   this->R = R;
18 }
19 void Circle::setC(const Point2D& C) {
20   this->C = C;
21 }
22 std::ostream& operator<<(std::ostream& o, const Circle& c) {
23   o << "Κύκλος με ακτίνα " << c.R << " και κέντρο " << c.C;
24   return o;
25 }
```

Σύνθεση (composition)

```cpp
#include <iostream>   //--- file: 08-testCircle.cpp
#include "08-Point2D.h"
#include "08-Circle.h"

int main() {
  Point2D p1;           // Σημείο με χρήση του default constructor
  Point2D p2(3.0, 4.0); // Σημείο με χρήση του constructor με παραμέτρους
  std::cout << "Σημείο p1: " << p1 << std::endl;   // Εμφάνιση των σημείων
  std::cout << "Σημείο p2: " << p2 << std::endl;

  p1.setx(1.5);    p1.sety(2.5);
  std::cout << "Νέες συντεταγμένες του p1: " << p1 << std::endl;

  Circle c1;                  // Κύκλος με χρήση του default constructor
  Circle c2(5.0, p2);         // Κύκλος με ακτίνα 5 και κέντρο p2
  Circle c3(10.0, 0.0, 0.0); // Κύκλος με ακτίνα 10 και κέντρο (0, 0)

  std::cout << "Κύκλος c1: " << c1 << std::endl;   // Εμφάνιση των κύκλων
  std::cout << "Κύκλος c2: " << c2 << std::endl;
  std::cout << "Κύκλος c3: " << c3 << std::endl;

  // Ρυθμίσεις για c1
  c1.setR(7.5);
  c1.setC(Point2D(2.0, 3.0));
  std::cout << "Νέες ιδιότητες του c1: " << c1 << std::endl;

  // Πρόσβαση και αλλαγή δεδομένων μέσω get
  double& radius = c2.getR();
  radius = 8.0; // Αλλαγή ακτίνας μέσω αναφοράς
  std::cout << "Αλλαγμένη ακτίνα του c2: " << c2.getR() << std::endl;
}
```

Κληρονομικότητα (inheritance)

Κληρονομικότητα στη C++

- Βασικός πυλώνας του αντικειμενοστραφούς προγραμματισμού.
- Επιτρέπει τη δημιουργία παράγωγων κλάσεων (*derived classes*) που βασίζονται σε υπάρχουσες (βασικές) κλάσεις (*base classes*), κληρονομώντας τις μεθόδους και τις ιδιότητες τους.
- Διευκολύνεται η **επανάχρηση του κώδικα**, η **επεκτασιμότητα** και η **συντήρηση**.

- **Base Class**: Η αρχική κλάση από την οποία κληρονομούν οι υπόλοιπες.
- **Derived Class**: Η νέα κλάση που κληρονομεί ιδιότητες και μεθόδους από την base class.

- **Είδη Κληρονομικότητας**:
  - **Public**: Οι *public/protected* μέθοδοι και ιδιότητες της *base class* διατηρούν την προσβασιμότητά τους στη *derived class*.
  - **Protected**: Οι *public* μέθοδοι γίνονται *protected* στη *derived class*.
  - **Private**: Όλες οι μέθοδοι και οι ιδιότητες γίνονται *private* στη *derived class*.

Κληρονομικότητα στη C++

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

class Base {
  public:
    void display() {
      cout << "Called the display() method from the Base class." << endl;
    }
};

class Derived : public Base {
  public:
    void show() {
      cout << "Called the show() method from the Derived class." << endl;
    }
};

int main() {
  Derived obj;
  obj.display(); // Κληρονομημένη μέθοδος από τη βασική κλάση
  obj.show();    // Μέθοδος της παράγωγης κλάσης
}
```

Υπερφόρτωση Κληρονομημένων Μελών στη C++

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

class Base {
  public:
    void print(int x) {
      cout << "Base class: x = " << x << endl;
    }
};

class Derived : public Base {
  public:
    void print(int x, int y) {
      cout << "Derived class: x = " << x << ", y = " << y << endl;
    }
    void print(double z) {
      cout << "Derived class: z = " << z << endl;
    }
};
int main() {
  Derived obj;

  // Χρήση υπερφορτωμένων μεθόδων της Derived class
  obj.print(10, 20);   // Κλήση με δύο ακέραιους
  obj.print(3.14);     // Κλήση με έναν δεκαδικό

  obj.print(5);           // Ποια μέθοδος θα κληθεί;...
  obj.Base::print(5);     // Άμεση κλήση της μεθόδου της Base class
  obj.Derived::print(5);  // Άμεση κλήση της μεθόδου της Derived class
}
```

Πολυμορφισμός (polymorphism)

Εικονικές Μέθοδοι και Πολυμορφισμός στη C++

- **Εικονικές Μέθοδοι (Virtual Methods)**: Οι εικονικές μέθοδοι επιτρέπουν την εκτέλεση της κατάλληλης υλοποίησης μίας μεθόδου σε μία ιεραρχία κλάσεων, βασιζόμενη στον τύπο του αντικειμένου κατά την εκτέλεση (runtime).
  - Δηλώνονται στη βασική κλάση με τη λέξη-κλειδί `virtual`.
  - Υλοποιούνται στις παράγωγες κλάσεις για εξειδικευμένη λειτουργικότητα.
  - Χρησιμοποιούνται μέσω δεικτών ή αναφορών στη βασική κλάση.
- **Καθαρά Εικονικές Μέθοδοι (Pure Virtual Methods)**: Μία **pure virtual method** είναι μία εικονική μέθοδος που δεν έχει υλοποίηση στη βασική κλάση. Δηλώνεται με την έκφραση `= 0` και καθιστά τη βασική κλάση αφηρημένη (*abstract class*). Οι παράγωγες κλάσεις πρέπει να υλοποιήσουν τη μέθοδο.
  - Χρησιμοποιείται για τη δημιουργία ιεραρχιών κλάσεων που λειτουργούν ως βάση για εξειδικεύσεις.
- **Πολυμορφισμός (Polymorphism)**: Η δυνατότητα χρήσης αντικειμένων διαφορετικών τύπων με ενιαίο τρόπο. Πραγματοποιείται μέσω:
  - Εικονικών μεθόδων.
  - Δυναμικής εκτέλεσης (dynamic dispatch): Όταν η μέθοδος καλείται μέσω δείκτη στη Base, εκτελείται η υλοποίηση της Derived.

Εικονικές Μέθοδοι και Πολυμορφισμός στη C++

Παράδειγμα:

```cpp
#include <iostream>
#include <cmath>
using namespace std;

class Shape {
  public:
    virtual double area() const = 0; // Καθαρά εικονική μέθοδος
};

class Circle : public Shape {
  private:
    double radius;
  public:
    Circle(double r) : radius(r) {}
    double area() const override {
      return M_PI * radius * radius; // A = πr²
    }
};

class Square : public Shape {
  private:
    double side;
  public:
    Square(double s) : side(s) {}
    double area() const override {
      return side * side; // A = α²
    }
};
```

Εικονικές Μέθοδοι και Πολυμορφισμός στη C++

Παράδειγμα (συνέχεια):

```
29 int main() {
30   Shape* shapes[4]; // Πίνακας τεσσάρων δεικτών προς αντικείμενα της κλάσης Shape
31   shapes[0] = new Circle(5.0); // Κύκλος με ακτίνα 5
32   shapes[1] = new Square(4.0); // Τετράγωνο με πλευρά 4
33   shapes[2] = new Circle(7.2); // Κύκλος με ακτίνα 7.2
34   shapes[3] = new Square(2.3); // Τετράγωνο με πλευρά 2.3
35
36   for (int i = 0; i < 4; ++i) {
37     cout << "Εμβαδόν: " << shapes[i]->area() << endl;
38   }
39
40   // Καθαρισμός μνήμης
41   delete shapes[0];
42   delete shapes[1];
43   delete shapes[2];
44   delete shapes[3];
45 }
```

- Δυναμική εκτέλεση (dynamic dispatch):
  Όταν η μέθοδος καλείται μέσω δείκτη στην base class (Shape),
  εκτελείται η υλοποίηση της derived class (Circle ή Square).

Abstract Classes στη C++

- **Αφηρημένη Κλάση (Abstract Class)**: Μία **abstract class** είναι μία κλάση που δεν μπορεί να δημιουργήσει αντικείμενα (instances). Χρησιμοποιείται για τον καθορισμό κοινής λειτουργικότητας και για τη δημιουργία ιεραρχιών κλάσεων.
- **Χαρακτηριστικά**:
  - Περιέχει τουλάχιστον μία **pure virtual method**, δηλαδή μέθοδο με τη δήλωση `= 0`.
  - Λειτουργεί ως βάση για παράγωγες κλάσεις που υλοποιούν τις pure virtual methods.
  - Δεν μπορεί να δημιουργηθεί αντικείμενο από αυτήν (π.χ. η εντολή `Shape s;` οδηγεί σε σφάλμα μεταγλώττισης).
- **Παράδειγμα Abstract Class**:

```
1 class Shape {
2 public:
3     virtual void draw() const = 0; // Pure virtual method
4     virtual ~Shape() {} // Εικονικός destructor
5 };
```

Concrete Classes στη C++

- **Συμπαγής Κλάση (Concrete Class)**: Μία **concrete class** είναι μία κανονική κλάση από την οποία μπορούν να δημιουργηθούν αντικείμενα (instances). Υλοποιεί όλες τις μεθόδους της, συμπεριλαμβανομένων αυτών που κληρονομεί.
- **Χαρακτηριστικά**:
  - Δεν περιέχει pure virtual methods.
  - Μπορεί να δημιουργήσει αντικείμενα και να χρησιμοποιηθεί κανονικά στο πρόγραμμα.
  - Εξειδικεύει ή επεκτείνει τη λειτουργικότητα των abstract classes (αν κληρονομεί από αυτές).
- **Παράδειγμα Concrete Class**:

```
1 class Circle : public Shape {
2 private:
3     double radius;
4 public:
5     Circle(double r) : radius(r) {}
6     void draw() const override {
7         cout << "Drawing a circle with radius " << radius << endl;
8     }
9 };
```