Αντικειμενοστραφής Προγραμματισμός

Μέρος 3ο: Εισαγωγή στην κληρονομικότητα

Εξάμηνο Σπουδών: 6ο

Κωδικός Μαθήματος: 647

Τμήμα Μαθηματικών
Πανεπιστήμιο Ιωαννίνων

Μιχάλης Α. Μπέκος
bekos@uoi.gr

Περιεχόμενα

Εισαγωγή στην κληρονομικότητα

- Αντικειμενοστρεφής προγραμματισμός
  - Ισχυρή προγραμματιστική τεχνική
  - Υποστηρίζει μια νέα διάσταση αφαιρετικότητας που ονομάζεται κληρονομικότητα
- **Ιδέα:** Ορίζεται μια γενική κλάση
  - στη συνέχεια, “εξειδικευμένες” κλάσεις κληρονομούν τις ιδιότητες της γενικής κλάσης,
  - οι οποίες επεκτείνονται με την προσθήκη ή τροποποίηση λειτουργιών (συναρτήσεων μελών)
- Πολύ χρήσιμο εργαλείο
  - Ουσιαστικό μέρος του αντικειμενοστραφή προγραμματισμού

Βασικά στοιχεία κληρονομικότητας

- Μία νέα κλάση (παράγωγη) κληρονομεί από άλλη κλάση (βασική)

- Βασική κλάση (ή υπερκλάση ή γονική κλάση):
  - είναι μια “γενική” κλάση από την οποία κληρονομούν άλλες

- Παράγωγη κλάση (ή υπόκλαση ή κλάση παιδί):
  - είναι μια νέα κλάση, η οποία έχει τα παρακάτω μέλη της βασικής κλάσης:
    - μεταβλητές μέλη
    - συναρτήσεις μέλη
  - μπορεί όμως να υποστηρίξει και επιπλέον μεταβλητές και συναρτήσεις μέλη.

Παράγωγες κλάσεις

- Παράδειγμα: Οι παρακάτω κλάσεις είναι ειδικές περιπτώσεις της Employee (Υπάλληλος):
  - HourlyEmployee (Ωρομίσθιος υπάλληλος)
  - SalariedEmployee (Μισθωτός υπάλληλος)
  - Μια άλλη κατηγορία είναι εκείνοι που πληρώνονται με σταθερό μισθό κάθε μήνα ή βδομάδα

```
class Employee {
public:
    Employee();
    string getName() const;
    string getSsn() const;
    double getNetPay() const;
    void printCheck() const;
private:
    string name, ssn;
    double netPay;
};
```

```
class HourlyEmployee : public Employee {
public:
    HourlyEmployee();
    double getRate() const;
    double getHours() const;
    void setRate(double newWageRate);
    void setHours(double hoursWorked);
    void printCheck();
private:
    double wageRate, hours;
};
```

Πίσω στο παράδειγμα...

- Στην πραγματικότητα, δεν χρειάζεται ο “γενικός τύπος”
  - αφού κανείς δεν είναι απλά ένας “υπάλληλος”
  - κάθε υπάλληλος εντάσεται σε ένα μισθωλογικό καθεστώς
- Αντικείμενα αυτής της κλάσης δεν θα δημιουργηθούν ποτέ.
- Ωστόσο, η έννοια του υπαλλήλου είναι χρήσιμη:
  - όλοι οι υπάλληλοι έχουν όνομα
  - όλοι έχουν αριθμό κοινωνικής ασφάλισης
  - οι λειτουργίες (συναρτήσεις μέλη) για αυτά τα “γενικά στοιχεία” είναι κοινές για όλους
- Άρα γιατί να πρέπει να υλοποιηθούν σε κάθε παράγωγη κλάση (code duplication);
  - η βασική κλάση περιέχει όλα αυτά τα “γενικά στοιχεία” για τους υπαλλήλους
  - και οι παράγωγες κλάσεις απλά τα κληρονομούν

Hands-on ενότητα 1

Υλοποίηση της βασικής κλάσης Employee

Η συνάρτηση μέλος `printCheck()` της κλάσης Employee

- Εκτυπώνει μια εντολή πληρωμής για τον υπάλληλο:
  - δεν έχει νόημα για ένα “γενικό” υπάλληλο
  - έτσι η `printCheck()` της κλάσης Employee απλά αναφέρει μήνυμα σφάλματος
- “Επαναπροσδιορίζεται” σε παράγωγες κλάσεις
  - διαφορετικοί τύποι εργαζομένων έχουν διαφορετικές εντολές πληρωμής

```
void Employee::printCheck() const {
    cout << "Error: printCheck function called for an" << endl
         << "undifferentiated employee. Aborting the program." << endl
         << "Check with the author of the program about this bug.";
    exit(1);
}
```

Κληρονομώντας από την κλάση Employee

- Παράγωγες κλάσεις της κλάσης Employee:
  - αυτόματα έχουν όλες τις μεταβλητές μέλη
  - και όλες τις συναρτήσεις μέλη της κλάσης Employee
- Οι παράγωγες κλάσεις λέγεται ότι κληρονομούν μέλη από τη βασική κλάση
- Μια παράγωγη κλάση μπορεί επίσης να:
  - επαναπροσδιορίσει τα μέλη που κληρονομεί
  - και/ή να υποστηρίξει νέα

Hands-on ενότητα 2

Υλοποίηση των παράγωγων κλάσεων HourlyEmployee και SalariedEmployee

Η κλάση HourlyEmployee

- Ο κώδικας της κλάσης ξεκινά όπως αυτός οποιοσδήποτε άλλης κλάσης:
  - Δομή `#ifndef`
  - Περίληψη των απαιτούμενων βιβλιοθηκών
  - Περίληψη επίσης της `employee.h`
- Ο ορισμός της κλάσης, ωστόσο, γίνεται διαφορετικά:
  - καθορίζει ότι "δημόσια κληρονομεί" από την κλάση Employee

```
#ifndef HOURLYEMPLOYEE_H
#define HOURLYEMPLOYEE_H

#include <string>
#include "employee.h"
using namespace std;

class HourlyEmployee : public Employee {
  //class code goes here
};
#endif //HOURLYMPLOYEE_H
```

Προσθήκες στην κλάση HourlyEmployee

- Η κλάση HourlyEmployee επιπρόσθετα υποστηρίζει:
  - Κατασκευαστές
  - Νέες μεταβλητές μέλη: `wageRate`, `hours`
  - Νέες συναρτήσεις μέλη: `setRate()`, `getRate()`, `setHours()`, `getHours()`
- Τα υπόλοιπα μέλη (`name`, `ssn`, `netPay`) ορίζονται κληρονομικά

```
class HourlyEmployee : public Employee {
public:
    HourlyEmployee();
    HourlyEmployee(string& theName, string& theSsn, double theWageRate, double theHours);
    void setRate(double newWageRate);
    double getRate() const;
    void setHours(double hoursWorked);
    double getHours() const;
    void printCheck();
private:
    double wageRate, hours;
};
```

Επαναπροσδιορισμοί στην κλάση HourlyEmployee

- Η κλάση HourlyEmployee επαναπροσδιορίζει:
  - τη συνάρτηση μέλος `printCheck()`
  - η υλοποίηση γίνεται ως μέρος της κλάσης HourlyEmployee
  - όμοια με τις υλοποιήσεις άλλων συναρτήσεων μελών της κλάσης HourlyEmployee
- Γίνεται δηλαδή "υπερσκελισμός" της συνάρτησης μέλους `printCheck()` της κλάσης Employee

```
void HourlyEmployee::printCheck() {
    setNetPay(hours * wageRate);
    cout << "_____________________________________________" << endl;
    cout << "Pay to the order of " << getName() << endl;
    cout << "The sum of " << getNetPay() << " Euros" << endl;
    cout << "_____________________________________________" << endl;
    cout << "Check Stub:  *NOT NEGOTIABLE*" << endl;
    cout << "Employee Number: " << getSsn() << endl;
    cout << "Hourly Employee." << endl << "Hours worked: " << hours
         << ", Rate: " << wageRate << ", Pay: " << getNetPay()  << endl;
    cout << "_____________________________________________" << endl;
}
```

Επαναπροσδιορισμός και υπερφόρτωση

- Ο επαναπροσδιορισμός σε μια παράγωγη κλάση:
  - ουσιαστικά “ξαναγράφει” την ίδια συνάρτηση (ίδια λίστα των παραμέτρων)
- Η υπερφόρτωση:
  - ορίζει μια “νέα” συνάρτηση με διαφορετικές παραμέτρους (διαφορετική υπογραφή)
- Σημαντική σημείωση: Όταν μια συνάρτηση μέλος επαναπροσδιορίζεται σε μια παράγωγη κλάση, ο ορισμός της στην βασική κλάση “δεν χάνεται”

```
int main {
    Employee JaneE;
    HourlyEmployee SallyH;
    JaneE.printCheck();              // Calls Employee's printCheck function
    SallyH.printCheck();             // Calls HourlyEmployee's printCheck function
    SallyH.Employee::printCheck(); // Calls Employee's printCheck function!
}
```

Κατασκευαστές σε παράγωγες κλάσεις

- Οι κατασκευαστές μιας βασικής κλάσης δεν κληρονομούνται σε παράγωγες κλάσεις
  - αλλά μπορούν να κληθούν από τον κατασκευαστή μιας παράγωγης κλάσης
  - για την αρχικοποιήση των μεταβλητών μελών που κληρονομούνται από τη βασική κλάση
  - ουσιαστικά είναι το “πρώτο πράγμα” που κάνει ο κατασκευαστής της παράγωγης τάξης

- Η κλήση γίνεται στο τμήμα αρχικοποίησης

```
HourlyEmployee::HourlyEmployee() : Employee(), wageRate(0), hours(0) {
    //deliberately empty
}

HourlyEmployee::HourlyEmployee(string&  theName, string& theNumber,
                               double theWageRate, double theHours)
    : Employee(theName, theNumber), wageRate(theWageRate), hours(theHours) {
    //deliberately empty
}
```

Μια σημαντική σημείωση

- Κάθε κατασκευαστής μιας παράγωγης κλάσης πρέπει πάντα να κάνει κλήση σε έναν από τους κατασκευαστές της βασικής κλάσης
- Διαφορετικά, ο προκαθορισμένος κατασκευαστής της βασικής κλάσης καλείται αυτόματα
- Παράδειγμα: Οι δύο παρακάτω υλοποιήσεις είναι ισοδύναμες
  - η πρώτη κάνει ρητή κλήση στον προκαθορισμένο κατασκευαστή της βασικής κλάσης
  - ενώ η δεύτερη όχι

```
HourlyEmployee::HourlyEmployee() : Employee(), wageRate(0), hours(0) {
    //deliberately empty
}
```

```
HourlyEmployee::HourlyEmployee() : wageRate(0), hours(0) {
    //deliberately empty
}
```

Ένα λεπτό σημείο στην κληρονομικότητα

- Μια παράγωγη κλάση κληρονομεί όλα τα ιδιωτικά μέλη της βασικής κλάσης
  - αλλά δεν έχει άμεση πρόσβαση σε αυτά
- Πρόσβαση στις ιδιωτικές μεταβλητές μέλη ή στις ιδωτικές συναρτήσεις μέλη έχουν μόνο οι συναρτήσεις μέλη της κλάσης στην οποία ορίζονται
  - Π.χ., μια συνάρτηση μέλος μιας παράγωγης κλάσης δεν μπορεί να καλέσει μια ιδιωτική συνάρτηση μέλος μιας βασικής κλάσης

```
void HourlyEmployee::printCheck() {
    netPay = hours * wageRate;  // error: there's no access to name, ssn, netPay
    cout << "_____________________________________________" << endl;
    cout << "Pay to the order of " << name << endl;
    cout << "The sum of " << netPay  << " Euros" << endl;
    cout << "_____________________________________________" << endl;
    cout << "Check Stub:  *NOT NEGOTIABLE*" << endl;
    cout << "Employee Number: " << ssn << endl;
    cout << "Hourly Employee." << endl << "Hours worked: " << hours
         << ", Rate: " << wageRate << ", Pay: " << getNetPay()  << endl;
    cout << "_____________________________________________" << endl;
}
```

Ένα λεπτό σημείο στην κληρονομικότητα

- Η πρόσβαση στις ιδιωτικές μεταβλητές μέλη είναι δυνατή έμμεσα μέσω των δημόσιων συναρτήσεων μελών (πρόσβασης ή μετάλλαξης) της βασικής κλάσης
  - οι ιδιωτικές συναρτήσεις μέλη απλά δεν είναι διαθέσιμες
  - και αυτό είναι λογικό καθώς
    - οι ιδιωτικές συναρτήσεις μέλη είναι απλά “υποστηρικτικές” συναρτήσεις
    - και χρησιμοποιούνται μόνο στην κλάση που έχουν οριστεί.

```
void HourlyEmployee::printCheck() {
    setNetPay(hours * wageRate);  // access name, ssn, netPay via member functions
    cout << "_____________________________________________" << endl;
    cout << "Pay to the order of " << getName() << endl;
    cout << "The sum of " << getNetPay() << " Euros" << endl;
    cout << "_____________________________________________" << endl;
    cout << "Check Stub:  *NOT NEGOTIABLE*" << endl;
    cout << "Employee Number: " << getSsn() << endl;
    cout << "Hourly Employee." << endl << "Hours worked: " << hours
         << ", Rate: " << wageRate << ", Pay: " << getNetPay()  << endl;
    cout << "_____________________________________________" << endl;
}
```

Ο προσδιοριστής protected

- Ορίζει ένα νέο επίπεδο κατηγοριοποίησης των μελών μιας κλάσης
- Επιτρέποντας την πρόσβαση σε παράγωγες κλάσεις
  - δηλαδή, σε συναρτήσεις μέλη παράγωγων κλάσεων
  - αλλά πουθενά αλλού (η πρόσβαση από άλλες κλάσεις δεν είναι εφικτή)
- Πιστεύεται ότι ίσως αυτό “παραβιάζει” την αρχή της απόκρυψης πληροφοριών

```
class Employee {
public:
  string getName() const;
  string getSsn() const;
  double getNetPay() const;
  void setName(string& n);
  void setSsn(string& s);
  void setNetPay(double p);
protected:
  string name, ssn;
  double netPay;
};
```

```
void HourlyEmployee::printCheck() {
  netPay = hours * wageRate;
  cout << "_____________________________________" << endl;
  cout << "Pay to the order of " << name << endl;
  cout << "The sum of " << netPay  << " Euros" << endl;
  cout << "_____________________________________" << endl;
  cout << "Check Stub:  *NOT NEGOTIABLE*" << endl;
  cout << "Employee Number: " << ssn << endl;
  cout << "Hourly Employee." << endl << "Hours worked: " << hours
       << ", Rate: " << wageRate << ", Pay: " << netPay  << endl;
  cout << "_____________________________________" << endl;
}
```

Συναρτήσεις που δεν κληρονομούνται

- Γενικός κανόνας: Όλες οι “κανονικές” συναρτήσεις μέλη μιας βασικής κλάσης κληρονομούνται σε παράγωγες κλάσεις

- Εξαιρέσεις:
  - οι κατασκευαστές (όπως έχουμε δει)
  - οι καταστροφείς (όπως θα δούμε)
  - οι αντιγραφείς
  - ο τελεστής ανάθεσης

Καταστροφείς σε παράγωγες κλάσεις

- Εάν ο καταστροφέας μιας βασικής κλάσης έχει υλοποιηθεί σωστά, τότε
  - είναι εύκολη η υλοποίηση ενός καταστροφέα σε παράγωγη κλάση
- Όταν καλείται ο καταστροφέας μιας παράγωγης κλάσης:
  - αυτόματα καλείται ο καταστροφέας της βασικής κλάσης
  - επομένως, δεν υπάρχει ανάγκη για ρητή κλήση
- Οι καταστροφείς παράγωγων κλάσεων χρειάζεται να αποδεσμεύσουν μόνο τις μεταβλητές μέλη της παράγωγης κλάσης
  - και τυχόν μεταβλητές στις οποίες “δείχνουν”

Η σειρά κλήσης των καταστροφέων

- Θεωρείστε το παρακάτω σενάριο:
  - η κλάση B είναι παράγωγη της κλάσης A
  - η κλάση C είναι παράγωγη της κλάσης B

- Όταν ένα αντικείμενο της κλάσης C βγαίνει εκτός πεδίου εφαρμογής:
  - ο καταστροφέας της κλάσης C καλείται πρώτος
  - ακολούθως, καλείται ο καταστροφέας της κλάσης B
  - τέλος, καλείται ο καταστροφέας της κλάσης A

- Η σειρά κλήσης δηλαδή είναι αντίθετη από αυτή των κατασκευαστών

Οι σχέσεις “είναι” και “έχει” ως μέρος του προγραμματισμού

- Η κληρονομικότητα ορίζει τη σχέση “είναι” μεταξύ των αντικειμένων
  - Π.χ., ένα αντικείμενο τύπου HourlyEmployee “είναι” και τύπου Employee

- Υπενθύμιση: Μια κλάση που περιέχει αντικείμενα μιας άλλης κλάσης ως δεδομένα μέλη ορίζει μια σχέση “έχει” μεταξύ αυτών
  - Π.χ., η κλάση LineSegment “έχει” δύο αντικείμενα της κλάσης Point ως δεδομένα μέλη (τα άκρα του ευθύγραμμου τμήματος)

Μέρος 3°:
Η κλάση Vector

Η κλάση vector

- Αποτελεί μέρος της τυπικής βιβλιοθήκης προτύπων (Standard Template Library STL)
  - “πίνακες” το μέγεθος των οποίων προσαρμόζεται κατά την εκτέλεση
- Σύνταξη αρχικοποίησης: `vector<type> variable_name;`
- Κάθε αντικείμενο τύπου `vector`:
  - Ορίζεται επί ενός τύπου (πρόκειται για κλάση πρότυπο)
  - Αποθηκεύει συλλογή τιμών αυτού του τύπου

```
int main() {
    vector<int> v;
    int next;
    do {
        cout << "Enter a number (negative to exit): ";
        cin >> next;
        v.push_back(next);
    } while (next > 0);
    print(v);
}
```

```
/**
 * Function to print vector's elements
 */
template <typename T>
void print(vector<T> v)
{
    for (int i=0; i<v.size(); i++)
        cout << v[i] << " ";
    cout << endl;
}
```

Η κλάση vector

- Η πρόσβαση στα στοιχεία του διανύσματος γίνεται μέσω του τελεστή `[]`
  - όπως ακριβώς με τους πίνακες
- Αλλά για την εισαγωγή στοιχείων συνίστανται:
  - η συνάρτηση μέλος `push_back`
- Η πρόσβαση στο πλήθος των στοιχείων του διανύσματος γίνεται με κλήση
  - της συνάρτησης μέλους `size()`

```
int main() {
    vector<int> v;
    int next;
    do {
        cout << "Enter a number (negative to exit): ";
        cin >> next;
        v.push_back(next);
    } while (next > 0);
    print(v);
}
```

```
/**
 * Function to print vector's elements
 */
template <typename T>
void print(vector<T> v)
{
    for (int i=0; i<v.size(); i++)
        cout << v[i] << " ";
    cout << endl;
}
```

Χωρητικότητα διανύσματος

- Η συνάρτηση μέλος `capacity()`
  - επιστρέφει τη χωρητικότητα του διανύσματος
  - δηλ, τις θέσεις μνήμης που έχουν εκχωρηθεί

- Η χωρητικότητα αυξάνεται ή ελαττώνεται αυτόματα ανάλογα με τις ανάγκες
  - τυπικά, η χωρητικότητα είναι μεγαλύτερη του μεγέθους

- Εάν η διαχείριση της μνήμης είναι κρίσιμη, η χωρητικότητα του διανύσματος μπορεί να οριστεί με μη αυτόματο τρόπο
  - `v.reserve(32);` `//sets capacity to 32`
  - `v.reserve(v.size()+10);` `//sets capacity to 10 more than size`

Μορφές κληρονομικότητας: Protected και Private

- Νέες “μορφές” κληρονομικότητας:
  - Protected: Τα δημόσια μέλη της βασικής κλάσης γίνονται protected στην παράγωγη κλάση
  - Private: Όλα τα μέλη της βασικής κλάσης γίνονται ιδιωτικά στην παράγωγη κλάση
- Και οι δύο χρησιμοποιούνται σπάνια
- Παραδείγματα:

```
// Protected inheritance
template <typename T>
class Queue<T> : protected deque<T> {
    // class code goes here
}
```

```
// Private inheritance
template <typename T>
class Stack<T> : private vector<T> {
    // class code goes here
}
```

Hands-on ενότητα 3

Υλοποίηση των δομών δεδομένων Stack και Queue

Πολλαπλή κληρονομικότητα

- Μια παράγωγη κλάση μπορεί να έχει περισσότερες από μία βασικές κλάσεις
- Οι περιπτώσεις ασάφειας είναι ατελείωτες
- Επικίνδυνο εγχείρημα:
  - ορισμένοι θεωρούν ότι δεν πρέπει ποτέ να χρησιμοποιείται
  - σίγουρα θα πρέπει να χρησιμοποιείται μόνο από έμπειρους προγραμματιστές
  - το πρόβλημα του διαμαντιού
- Παράδειγμα:

```
// Multiple inheritance
class Professor : public Employee, FacultyMember {
    // class code goes here
}

// Assumption: print() is inherited from both classes
```

```
int main () {
    Professor p;
    p.Employee::print();
    p.FacultyMember::print();
    p.print(); // <-- ambiguous call
}
```

Σύνοψη

- Η κληρονομικότητα υποστηρίζει την επαναχρησιμοποίηση κώδικα
  - επιτρέπει σε μια κλάση να “προσδιορίζεται” μερικώς από μια άλλη,
  - και ταυτόχρονα να έχει επιπλέον χαρακτηριστικά

- Μια παραγόμενη κλάση κληρονομεί τα μέλη της βασικής κλάσης
  - υποστηρίζοντας παράλληλα νέα μέλη

- Οι παράγωγες κλάσεις δεν έχουν πρόσβαση σε ιδιωτικά μέλη βασικών κλάσεων
  - μια παράγωγη κλάση όμως έχει πρόσβαση στα protected μέλη μιας βασικής κλάσης

- Κατασκευαστές και καταστροφείς δεν κληρονομούνται στις παράγωγες κλάσεις

- Η κλάση `vector`: “πίνακες” το μέγεθος των οποίων προσαρμόζεται κατά την εκτέλεση