ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΑΤΡΩΝ
UNIVERSITY OF PATRAS

ΑΝΟΙΚΤΑ ακαδημαϊκά μαθήματα ΠΠ

Οντοκεντρικός Προγραμματισμός

Ενότητα 6: C++ ΚΛΑΣΕΙΣ, ΚΛΗΡΟΝΟΜΙΚΟΤΗΤΑ, ΠΟΛΥΜΟΡΦΙΣΜΟΣ

Πολυμορφισμός

ΔΙΔΑΣΚΟΝΤΕΣ:Ιωάννης Χατζηλυγερούδης, Χρήστος Μακρής

Πολυτεχνική Σχολή

Τμήμα Μηχανικών Η/Υ & Πληροφορικής

Πολυμορφισμός

Εισαγωγή

- Πολυμορφισμός
  - Απαιτείται η ύπαρξη μιας ιεραρχίας κλάσεων
  - Χρησιμοποιούμε αντικείμενα που ανήκουν στην ίδια ιεραρχία κλάσεων ως να ήταν όλα αντικείμενα της κλάσης βάσης
  - Εικονικές συναρτήσεις + δυναμική σύνδεση (Virtual functions + dynamic binding)
    - Τεχνολογίες που υλοποιούν τον πολυμορφισμό
  - Επιτρέπει την αποτελεσματική επέκταση των προγραμμάτων
    - Νέες κλάσεις μπορούν εύκολα να προστεθούν και να υποστούν επεξεργασία όπως και οι υπάρχουσες

Παράδειγμα

- Κλάση βάσης: *Animal*
- Παραγόμενες κλάσεις: *Fish*, *Frog*, *Bird*
- Η μέθοδος *move()* είναι κοινή, αλλά υλοποιείται διαφορετικά για κάθε είδος *Animal*.
- Μια εφαρμογή κρατά έναν πίνακα από αντικείμενα των παραγόμενων κλάσεων και κάθε 1sec στέλνει ένα μήνυμα *move* σε κάθε αντικείμενο.
- Κάθε αντικείμενο τύπου *Animal* που δέχεται το μήνυμα γνωρίζει πως θα πρέπει να αντιδράσει.
- Το ίδιο μήνυμα που στέλνεται σε διαφορετικά αντικείμενα (του ίδιου βασικού τύπου *Animal*) έχει πολλές μορφές αποτελεσμάτων → πολυμορφισμός
- Πόσο εύκολο είναι να προσθέσουμε μια νέα κλάση *Turtle*?

Παράδειγμα

```
class Animal{
  public:
    Animal(char* n):name(n){}
    void print() const{
      cout << "--------------------\n";
      cout << name << " is an animal";
    }
  private:
    char* name;
};
class Dog : public Animal{
  public:
    Dog(char* n):Animal(n){}
    void print() const{
      Animal::print();
      cout << ", and a dog." << endl; }
};
class Cat : public Animal{
  public:
    Cat(char* n):Animal(n){}
    void print() const{
      Animal::print();
      cout << ", and a cat." << endl;
    }
};
class Bird : public Animal{
  public:
    Bird(char* n):Animal(n){}
    void print() const{
      Animal::print();
      cout << ", and a bird." << endl;
    }
};
```

Υπερκάλυψη της μεθόδου print στις παράγωγες κλάσεις

```
int main() {
  Dog d("Pluto");
  d.print();
  Cat c("Tom");
  c.print();
  Bird b("Twitty");
  b.print();

  Animal *animalPointer = &d;
  animalPointer->print();
}
```

Ανάθεση αντικειμένου παράγωγης κλάσης σε δείκτη της κλάσης βάσης.

```
--------------------
Pluto is an animal, and a dog.
--------------------
Tom is an animal, and a cat.
--------------------
Twitty is an animal, and a bird.
--------------------
Pluto is an animal
```

Η μέθοδος που θα κληθεί εξαρτάται από την κλάση του δείκτη, (εδώ είναι τύπου Animal)

Διαφορετική συμπεριφορά από την Java

Παράδειγμα - virtual

```
class Animal{
  public:
    Animal(char* n):name(n){}
    virtual void print() const{
      cout << "------------------\n";
      cout << name << " is an animal";
    }
  private:
    char* name;
};
class Dog : public Animal{
  public:
    Dog(char* n):Animal(n){}
    void print() const{
      Animal::print();
      cout << ", and a dog." << endl; }
};
class Cat : public Animal{
  public:
    Cat(char* n):Animal(n){}
    void print() const{
      Animal::print();
      cout << ", and a cat." << endl;
    }
};
class Bird : public Animal{
  public:
    Bird(char* n):Animal(n){}
    void print() const{
      Animal::print();
      cout << ", and a bird." << endl;
    }
};
```

Αλλαγή της προκαθορισμένης συμπεριφοράς με virtual δήλωση

```
int main() {
  Dog d("Pluto");
  d.print();
  Cat c("Tom");
  c.print();
  Bird b("Twitty");
  b.print();

  Animal *animalPointer = &d;
  animalPointer->print();
}
```

```
------------------
Pluto is an animal, and a dog.
------------------
Tom is an animal, and a cat.
------------------
Twitty is an animal, and a bird.
------------------
Pluto is an animal, and a dog.
```

Η μέθοδος που θα κληθεί εξαρτάται από την πραγματική κλάση του αντικειμένου (δημιουργήθηκε ως Dog)

Δείκτης παραγόμενης κλάσης σε αντικείμενο της κλάσης βάσης

- Στο προηγούμενο παράδειγμα είδαμε
  - Δείκτης κλάσης βάσης σε αντικείμενο της παραγόμενης κλάσης
    - **`Dog is an Animal`**
- Δείκτης παραγόμενης κλάσης σε αντικείμενο της κλάσης βάσης
  - Compiler error
    - **`Animal`** is not a **`Dog`**
    - Η κλάση **`Dog`** μπορεί να διαθέτει ιδιότητες/λειτουργίες που δεν διαθέτει η κλάση **`Animal`**
  - Μπορούμε να κάνουμε cast τη διεύθυνση του αντικειμένου της κλάσης βάσης σε δείκτη παραγόμενης κλάσης
    - Ονομάζεται downcasting
    - Επιτρέπει την προσπέλαση λειτουργικότητας της παραγόμενης κλάσης

```
int main() {
  Animal a("Unknown");
  Dog d("Pluto");

  Animal* animalPointer = &d;
  Dog* dogPointer = &a;
}
```

Κλήση μεθόδων παραγόμενης κλάσης μέσω δείκτη κλάσης βάσης

- Χρήση δείκτη/αναφοράς
  - Δείκτης κλάσης βάσης μπορεί να δείχνει σε αντικείμενο της παραγομένης κλάσης
    - Μπορεί όμως να καλέσει μόνο τις μεθόδους της κλάσης βάσης
  - Η κλήση μεθόδων της παραγόμενης κλάσης συνιστά λάθος
    - Μέθοδοι που δεν ορίζονται στη κλάση βάσης
- Ο τύπος δεδομένων ενός/μιας δείκτη/αναφοράς προσδιορίζουν τι μεθόδους μπορούμε να καλέσουμε

Εικονικές Συναρτήσεις
(Virtual Functions)

- Ο κανόνας είναι ο τύπος του δείκτη να καθορίζει τι μέθοδοι μπορούν να κληθούν
- Οι **`virtual`** functions μπορούν να τον αλλάξουν
  - το αντικείμενο (και όχι ο δείκτης) να καθορίζει τι μέθοδοι μπορούν να κληθούν

Εικονικές Συναρτήσεις (Virtual Functions)

- Δηλώνουμε την **`print`** ως **`virtual`** στην κλάση βάσης
  - Υπερκαλύπτουμε τη **`print`** σε κάθε παραγόμενη κλάση
    - σαν να την ξαναορίζουμε, αλλά ή νέα συνάρτηση πρέπει να έχει ακριβώς την ίδια υπογραφή με αυτή της κλάσης βάσης
  - Εάν δηλώσουμε μια συνάρτηση **`virtual`** στην κλάση βάσης
    - **`virtual void print() const;`**
    - είναι **`virtual`** σε όλες τις παραγόμενες κλάσεις
      - Είναι καλή πρακτική να δηλώνουμε ρητά ως **`virtual`** τις συναρτήσεις και στις παραγόμενες κλάσεις (program clarity)
- Δυναμική σύνδεση (Dynamic binding)
  - Επιλογή κλήσης της κατάλληλης συνάρτησης κατά το χρόνο εκτέλεσης
  - Συμβαίνει μόνο με τη χρήση δεικτών/αναφορών
    - Εάν η συνάρτηση κληθεί με χρήση ονόματος αντικειμένου (π.χ. **`dogObject.print()`**), τότε χρησιμοποιείται η συνάρτηση του αντικειμένου (static binding)

Χρήση πεδίου για τύπο αντικειμένων και εντολής switch

- Ένας τρόπος να προσδιορίσουμε την κλάση ενός αντικειμένου
  - Βάζουμε στην κλάση βάσης ένα πεδίο
    - **`shapeType`** στην κλάση **`Shape`**
  - Χρησιμοποιούμε μια **`switch`** για να καλέσουμε τη σωστή **`print`** συνάρτηση
- Πολλά προβλήματα
  - Μπορεί να ξεχάσουμε τον έλεγχο κάποιας περίπτωσης στην **`switch`**
  - Εάν προσθέσουμε/αφαιρέσουμε μια κλάση πρέπει να ενημερώσουμε την **`switch`**
    - Χρονοβόρο και επιρρεπές σε λάθη
- Καλύτερα με πολυμορφισμό
  - απλούστερα προγράμματα, λιγότερο debugging

Abstract Classes

- Abstract class
  - Σκοπός: να αποτελέσει μια base class (abstract base class)
  - Ελλιπής
    - Συμπληρώνεται με τις παραγόμενες κλάσεις
  - Δεν κατασκευάζουμε αντικείμενα από μια abstract class
    - Μπορούμε να έχουμε δείκτες και αναφορές
- Concrete classes
  - Μπορούμε να κατασκευάσουμε αντικείμενα
  - Υλοποιούμε όλες τις συναρτήσεις που περιέχουν

Abstract Classes

- Είναι χρήσιμες, όχι υποχρεωτικές
- Για να ορίσουμε μια abstract class
  - Δεν υπάρχει το keyword abstract όπως στην Java
  - Θέλουμε μια ή περισσότερες "pure" virtual functions

```
virtual void print() const = 0;
```

  - Regular virtual functions
    - Έχουν υλοποίηση, η υπερκάλυψη είναι προαιρετική
  - Pure virtual functions
    - Δεν έχουν υλοποίηση, η υπερκάλυψη είναι υποχρεωτική
  - Μια abstract class μπορεί να έχει δεδομένα και υλοποιημένες συναρτήσεις

Παράδειγμα

```
#include <vector>
using std::vector;
…

void virtualViaReference( const Animal &baseClassRef ){
   baseClassRef.print();
}
void virtualViaPointer( const Animal *baseClassPtr ){
   baseClassPtr->print();
}
int main() {
   Animal a("Unknown");
   Dog d("Pluto");
   Cat c("Tom");
   Bird b("Twitty");
   vector< Animal* > animalVector( 3 );
   animalVector [ 0 ] = &d;
   animalVector [ 1 ] = &c;
   animalVector [ 2 ] = &b;
   cout << " VIRTUAL VIA REFERENCE" <<endl;
   for ( int i = 0; i < animalVector.size(); i++ )
     virtualViaReference( *animalVector[ i ] );
   cout <<endl<<endl<< " VIRTUAL VIA POINTER" <<endl;
   for ( int i = 0; i < animalVector.size(); i++ )
     virtualViaPointer( animalVector[ i ] );
}
```

Virtual Destructors

- Δείκτης κλάσης βάσης σε αντικείμενο της παραγόμενης κλάσης
  - Εάν καταστρέψουμε το αντικείμενο με την **`delete`**, η συμπεριφορά είναι απροσδιόριστη
- Απλή λύση
  - Δηλώνουμε τον destructor της base-class ως virtual
  - Όταν καλείται η **`delete`** καλείται και ο κατάλληλος destructor
- Όταν καταστρέφουμε ένα αντικείμενο μιας παραγόμενης κλάσης
  - Πρώτα εκτελείται ο destructor της παραγόμενης κλάσης
  - Μετά εκτελείται ο destructor της base-class
- Οι constructors δεν μπορεί να είναι virtual

Παράδειγμα

```
#include <vector>
#include <typeinfo>
using std::vector;
…

int main() {
  vector< Animal* > animalVector( 3 );
  animalVector [ 0 ] = new Dog("Pluto");
  animalVector [ 1 ] = new Cat("Tom");
  animalVector [ 2 ] = new Bird("Twitty");

  for ( int i = 0; i < animalVector.size(); i++ ){
    Dog *dogPtr = dynamic_cast <Dog *>( animalVector[ i ] );
    if (dogPtr) animalVector[i]->print();
  }

  for ( int i = 0; i < animalVector.size(); i++ ){
    cout <<endl<< "deleting object " << typeid (*animalVector[ i ]).name() ;
    delete animalVector[i];
  }
}
```

Η **dynamic_cast** επιστρέφει null αν δεν μπορεί να γίνει το casting

Η **typeid** επιστρέφει αντικείμενο type_info, το οποίο περιέχει πληροφορίες όπως το όνομα του αντικειμένου με την μέθοδο name()

```
------------------
Pluto is an animal, and a dog.

deleting object 3Dog
deleting object 3Cat
deleting object 4Bird
```

Πρόσθετο Υλικό

- Μελετήστε και τα παραδείγματα από τα **Κεφάλαιο 12** του βιβλίου:
  «C++ How to Program, 9/e Paul & Harvey Deitel»
  http://media.pearsoncmg.com/ph/esm/deitel/cpp_htp_9/code_examples/Code_Examples.zip

Χρηματοδότηση

- Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στο πλαίσιο του εκπαιδευτικού έργου του διδάσκοντα.
- Το έργο «**Ανοικτά Ακαδημαϊκά Μαθήματα στο Πανεπιστήμιο Αθηνών**» έχει χρηματοδοτήσει μόνο την 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.

Σημείωμα Ιστορικού Εκδόσεων Έργου

Το παρόν έργο αποτελεί την έκδοση 1.0.1

Σημείωμα Αναφοράς

Σημείωμα Αδειοδότησης

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.

Σημείωμα Χρήσης Έργων Τρίτων

- Οι διαφάνειες βασίζονται στο βιβλίο «C++ How to Program, 8th Edition, Harvey M. Deitel, Paul J. Deitel, Prentice Hall.»