МАТЕМАТИКА И МАТЕМАТИЧЕСКО ОБРАЗОВАНИЕ, 2008
MATHEMATICS AND EDUCATION IN MATHEMATICS, 2008
Proceedings of the Thirty Seventh Spring Conference of
the Union of Bulgarian Mathematicians
Borovetz, April 2–6, 2008

ПОДХОД ЗА УСВОЯВАНЕ НА КОНЦЕПЦИИТЕ НА ООП ЧРЕЗ ПРИМЕРИ

Ивайло Петров Дончев

Обектно ориентираното програмиране (ООП) е сред най-влиятелните методи за програмиране в наши дни. Във всеки учебен план на информатична специалност неизменно присъства дисциплината ООП. Актуален е въпросът каква методология е най-подходяща за обучението по тази дисциплина. В този доклад се предлага методология за усвояване на основните понятия и механизми на ООП по време на лабораторни занятия, базирана на индуктивен подход, с плавен преход от процедурен към обектно ориентиран стил на програмиране. Ключова роля в нея има постепенното обогатяване на примерните класове с нови възможности и тяхната оптимизация, реализирани чрез прилагане на нововъведените концепции.

1. Въведение. Въпреки натрупания вече десетилетен опит, обучението по програмиране и в частност по ООП си остава трудна задача [2,5,7,14]. Студентите срещат трудности с основните понятия клас и обект, връзките между класовете и тяхното взаимодействие, използването на паметта, комплексните обекти, наследяването и полиморфизма, цялостната картина на работа на програмата [13]. Много изследвания са посветени на търсене на причините за тези трудности. Няма единно мнение по въпроса. Най-често цитирана причина е преминаването от процедурен към обектно ориентиран стил на програмиране (paradigm shift) [11,7]. "Превключването" е трудно, а не самото ООП [7]. За друга немаловажна причина се сочи работата с тежките професионални програмни среди и липсата на добри помощни дидактически средства [8,10].

За ограничаване на срещаните трудности е важно още в уводните курсове по програмиране да се развиват способностите за абстрактно мислене и моделиране у студентите. Добра предпоставка за това е и фактът, че курсът "Алгоритми и структури от данни" в повечето учебни планове е включен преди ООП.

Много изследвания сочат, че студентите предпочитат да учат чрез примери [4,9]. Мнозинството от хората учат по-лесно, ако в началото им се дават частни примери, а след това се прави обобщение [3] (индуктивен подход). Това с особена сила важи за обучението по програмиране. Ето защо е съществен въпросът какви примери и задачи са подходящи за успешното усвояване на концепциите на ООП.

В този доклад предлагам методология, при която на няколко стъпки, чрез разширяване на едни и същи примерни класове, се усвояват новите концепции. Използват се примери, описващи ситуации от реалния живот или такива, решаващи проблеми от математиката (сериозно застъпена в учебните програми за информатици).

Такива са класическите примери с моделиране на данни за хора - `class Person`, в последствие разширяван до `class Student`; за автомобили – `Vehicle - Car, Truck`; геометрични фигури – `Point - Line, Rectangle, Circle` и др. С такива примери могат да се обхванат всички важни концепции на ООП – абстрактни типове данни, капсулиране, наследяване, полиморфизъм.

За да се намали отрицателното влияние от смяната на програмната парадигма, е важно тези примери да са разглеждани и решавани алгоритмично още в уводния курс по програмиране, първо по процедурен начин. Акцентът при ООП вече не е върху алгоритъма и структурите данни, а върху създаването на ефективен модел и система от класове, представящи ситуацията. В курса по ООП същите решени в процедурен стил задачи се моделират в класове и обекти чрез средствата на новата парадигма.

За примерите е използван хибридният език C++. Този избор е сполучлив, тъй като по учебен план студентите първо изучават процедурните конструкции, след което преминават към обектно ориентирания материал [1]. Така, започвайки обучението си по ООП, студентите вече са усвоили синтаксиса на езика и са свикнали със средата за разработка.

Предлаганата методология включва няколко последователни стъпки, като всяка следваща изисква прилагането на нова ООП концепция:

2. Упражняване на синтаксиса на езика. На студентите се поставя задача, която те трябва да решат със средствата на процедурното програмиране (това, което вече са овладели в предишните курсове), например, да се състави програма, която поддържа информация за хора – име, година на раждане, адрес. Програмата да въвежда данни за 10 човека в някаква структура (масив) и да предлага възможност за търсене по някакъв критерий. Задачата не е нова за студентите и не се очакват трудности при решаването и. Предложените решения трябва да съдържат следните елементи:

```
struct Person {              // структура, описваща данните за човек
char name[25];               // име
int year;                    // година на раждане
char address[50];            // адрес
};
void getData(Person&);       // функция за въвеждане на данни
void display(Person);        // функция за извеждане на данни
void search();               // функция, реализираща търсенето
```

Тази стъпка е необходима, за да си припомнят студентите синтаксиса на езика и работата със средата.

3. Усвояване на концепциите клас и обект. След уводна лекция, в която студентите трябва да възприемат основните понятия клас и обект, се преминава към втората стъпка. Именно тук е сблъсъкът с трудността на прехода от процедурно програмиране към ООП. Програмата трябва да се преработи така, че да удовлетворява изискванията на новата парадигма – за момента това са класове, обекти и капсулиране на данните. В резултат се получава следния клас:

```
class Person {            // клас
      char name[25];
      int year;
      char address[50];
public:
      void getData();     // метод за въвеждане на данни
      void display();     // метод за извеждане на данни
};
```

Тук студентите трябва да се запознаят със синтаксиса на деклариране на класове и дефиниране на методи, да се изясни разликата между функция (процедура) и метод, както и между клас и обект, да се коментира въпросът с достъпа до компонентите на класа и тяхната "видимост", да се изясни ролята на указателя `this`. Демонстрира се инстанцирането, използването на масиви от обекти и указатели към обекти.

4. Конструиране и разрушаване на обекти. Студентите срещат трудности при работа с паметта – заделяне, освобождаване и достъп до динамична памет, използване на указатели и псевдоними. Ако не се преодолеят тези трудности, става невъзможно разбирането на цялостната работа на програмата [15]. Затова на конструирането на обекти трябва да се отдели специално внимание.

Разглежданата примерна програма би могла да се оптимизира по отношение на използваната памет като се заменят масивите от символи с указатели и инициализацията на компонентите се възложи на подходящи конструктори:

```
class Person {
      char *name;                 // Указател вместо масив
      int year;
      char *address;              // Указател вместо масив
public:
      Person();                   // Подразбиращ се конструктор
      Person(char*, int, char*);  // Конструктор с параметри
      Person(Person&);            // Копиращ конструктор
      ~Person();                  // Деструктор
      void display();             // Метод за извеждане на данни
};
```

С примера се демонстрира използването на статичен буфер в рамките на конструктора, динамично заделяне и освобождаване на памет, инициализация чрез друг обект (с копиращ конструктор), предефинирани конструктори.

5. Наследяване. Механизмът на наследяване е основен за ООП, което много често се определя като АТД + наследяване + полиморфизъм. Наследяването е важна техника за реализиране на концепциите за повторна използваемост на код (reusability) и разширяемост (extendibility). Самият процес се състои в създаване на нови класове, наречени производни, от вече създадени класове, наречени базови. Езикът C++ ефективно поддържа всички възможности на маханизма на наследяване, включително множествено наследяване, и е особено подходящ за демонстрирането му.

Примерният клас `Person` лесно може да се разшири с производен клас `Student`. Така се демонстрира просто наследяване:

```
class Student : public Person {            // производен клас
      char* f_number;                      // факултетен номер
      int course;                          // курс
      ..................................   // други данни
public:
      Student();                           // подразбиращ се конструктор
      Student(char*,int,char*,char*,int);  // с параметри
      ~Student();                          // деструктор
      void display();                      // едноименен метод
      ..................................   // други методи
};
```

С този пример се демонстрират: дефиниране на производен клас, достъп до наследени компоненти на производни класове, конструктори и деструктори на производни класове, едноименни методи в производни и базови класове.

Механизмът на множествено наследяване може да се покаже с обединяването на два класа от вече разгледани примери. За нуждите на обучението йерархията от класове не трябва да бъде много сложна. Например, от базовите класове `Person` и `Car` да се получи производният клас `CarPassport`:

```
class CarPassport : public Person, public Car {
public:
      CarPassport(char*, int, char*, char*, char*, int);
      void display();
      ..............................
};
```

6. Късно свързване и полиморфизъм. В програмирането полиморфизмът означава възможност за дефиниране на един и същ метод за обекти на различни класове. Методът ще извършва различни действия, в зависимост от типа на обекта, върху който се прилага. В езиците за ООП полиморфизмът се реализира чрез динамично свързване, виртуални функции, абстрактни класове. Динамичното свързване позволява различни обекти в системата да отговарят на едно и също съобщение по специфичен за всеки тип начин. Получателят на съобщението се определя динамично по време на работата на програмата.

В C++, за да се реализира едно полиморфично действие, класовете на обектите, върху които то ще се прилага трябва да имат общ корен (родител или прародител), т.е. да бъдат производни на един един и същ клас. В този клас трябва да бъде дефиниран виртуален метод, съответстващ на полиморфичното действие. Във всеки от производните класове методът трябва да бъде предефиниран, съобразно особеностите на конкретния клас. Активирането на полиморфичното действие става чрез указател (или псевдоним) към базовия клас, на който могат да се присвояват адреси

на обекти на производните класове. Благодарение на механизма на виртуалните методи, ще бъде изпълняван методът на обекта, към който сочи указателят в момента на извикването.

Удачно е да се демонстрира полиморфизмът с едно негово характерно приложение – създаване на хетерогенни структури от данни (списъци, дървета, графи и др., чиито елементи са обекти на различни класове). Това може да стане с използване на познатите от предишните примери класове `Person` и `Car`, за да се създаде едносвързан списък с елементи обекти на тези два класа. За полиморфично действие може да се избере извеждането на данните за всеки от елементите на списъка, реализирано от виртуалния метод `display()`.

```
class Node {           // базов абстрактен клас - възел от списъка
public:
      Node* next;      // указател към следващ елемент от списъка
      virtual void display()=0; // чист виртуален метод
};
class Person : public Node {     // производен клас
      ........................   // за единия тип елементи
      virtual void display();    // предефиниране на виртуалния метод
      ........................   // реализиращ полиморфичното действие
};
class Car : public Node {        // производен клас
      ........................   // за другия тип елементи
      virtual void display();    // предефиниране на виртуалния метод
      ........................   // реализиращ полиморфичното действие
};
class List {                     // списък
      Node *top;                 // указател към абстрактния клас
public:
      List();                    // конструктор
      ~List();                   // деструктор
      void Add();                // добавяне на елемент от списъка
      void display();            // извеждане на целия списък
};
```

Резултатът от полиморфизма се демонстрира от метода `display()` на класа `List`. В зависимост от типа на обекта, записан в съответния възел от списъка, се извиква подходящият мотод `display()` на класа `Person` или `Car`, като изборът се прави по време на изпълнението на програмата:

```
void List::display()
{
     for(Node* p = top; p; p = p → next)
     p → display();
}
```

6. Резултати от анкетата. За проверка на това как студентите възприемат идеите на ООП и дава ли предложената методология резултати, проведох анкета с тях. Резултатите сочат следното:

• За 74% от анкетираните ООП ще бъде предпочитан в бъдещата им работа;

• Опитът от процедурното програмиране помага в обучението по ООП според 76% от анкетираните;

• 88% смятат, че наученото в курса по ООП им помага в другите изучавани дисциплини;

• 56% предпочитат в учебния курс примери от "реалния живот", а 12% – чисто математически;

• Само 21% са срещнали големи трудности при усвояването на новите концепции.

7. Заключение. Преподавателите по компютърни науки използват много примери, които могат да нанесат повече вреда, отколкото полза [6,12]. Затова е задължително примерите да са не само коректни, но също така да служат като шаблон за добро проектиране и стил във всеки един аспект. Това изисква тяхното внимателно планиране и разработване. Опитът сочи, че примерите, конструирани "в движение" почти сигурно внасят неумишлени проблеми и грешки. Мое убеждение е, че най-подходящи за ООП са сравнително простите примери с обем до няколко страници код, но такива, че да илюстрират силата на обектно ориентираните идеи. Класовете да бъдат с богати възможности, но лесни за използване и разширяване. Важно е също изграждането на умерено сложни йерархии от добре взаимодействащи си класове. Усвояването на концепциите се улеснява, ако се моделират едни и същи реални ситуации, като примерните класовете постепенно се разширяват и оптимизират с въвеждането на новите знания и средства.

Подготовката на студентите за усвояване на концепциите на ООП трябва да започне още в уводните курсове по програмиране с развитие на способностите им за алгоритмично и абстрактно мислене, моделиране на реални ситуации и усвояване на синтаксиса и семантиката на хибриден език за програмиране.

ЛИТЕРАТУРА

[1] C. Alphonce. "Object orientation in CS1-CS2 by design" , Proceedings of the 7th annual conference on Innovation and technology in computer science education, Aarhus, Denmark, 2002, Pages: 70–74.

[2] J. Börstler, H. Sharp. "Learning and teaching object technology", *Editorial Computer Science Education*, **13** (2003), No 4, 243–247.

[3] K. Bruce. "Controversy on how to teach CS 1: a discussion on the SIGCSE-members mailing list", *ACM SIGCSE Bulletin*, **37**, Issue 2 (June 2005), 111–117, ACM Press New York, NY, USA.

[4] M. Chi, M. Bassok, M. Lewis, P. Reimannq R. Glasser. "Self-explanations: How students study and use examples in learning to solve problems", Cognitive Science, **13** (1989), 145–182.

[5] A. Ebrahimi, C. Schweikert. "Empirical study of novice programming with plans and objects", Working group reports on ITiCSE on Innovation and technology in computer science education, Bologna, Italy, pp. 52–54, 2006.

[6] C. Hu. "Dataless objects considered harmful", *Commun. ACM*, **48** (2005), 99–101.

[7] M. Kölling. "The Problem of Teaching Object-Oriented Programming, Part 1: Languages", *Journal of Object-Oriented Programming*, **11** (1999), No 8, 8–15.

[8] M. Kölling. "The Problem of Teaching Object-Oriented Programming, Part 2: Environments", *Journal of Object-Oriented Programming*, **11** (1999), No 9, 6–12.
[9] J. LeFevre, P. Dixon. "Do Written Instructions Need Examples?", *Cognition & Instruction*, **3** (1986), pp. 1.
[10] S. Leung. "Integrating Visualization to Make Programming Concepts Concrete – Dot Net Style", *Proceedings of the 7th conference on Information technology education*, Minneapolis, Minnesota, USA, 2006, 149–156.
[11] C. Liu, S. Goetze, B. Glynn. "What Contributes to Successful Object-OrientedLearning?", OOPSLA '92 Proceedings., pp. 77 – 86, 1992
[12] K. Malan, K. Halland. "Examples that can do harm in learning programming", in OOPSLA '04: *Companion to the 19th Annual ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages, and Applications*, 2004, pp. 83–87.
[13] V. Ramesh, Wu Ji-Tsung Ben. "An Approach to Teaching Object-Oriented Programming Concepts in Business Schools", *Decision Sciences Journal of Innovative Education*, **2** (2004), No 1, pp. 83–87.
[14] A. Robins, J. Rountree. "Learning and Teaching Programming: A Review and Discussion", *Journal of Computer Science Education*, **13** (2003), pp. 137–172.
[15] C. Schulte, J. Bennedsen. "What do teachers teach in introductory programming?", *Proceedings of the 2006 international workshop on Computing education research*, Canterbury, United Kingdom, 17–28.

ВТУ „Св. Св. Кирил и Методий"
Педагогически факултет
катедра „Информационни технологии"
ул „арх. Г. Козарев" № 3
5000 Велико Търново
e-mail: i.donchev@abv.bg

AN APPROACH TO TEACHING OBJECT-ORIENTED PROGRAMMING CONCEPTS BY EXAMPLES

Ivaylo Petrov Donchev

Object-oriented programming (OOP) is among the most influential programming paradigms nowadays. Each IT curriculum invariably includes OOP as a subject. A question of present interest is what is the most suitable instructional methodology in this subject. The present paper proposes a methodology for teaching the basic concepts and mechanisms of OOP during laboratory classes based on an inductive approach, with a smooth transition from a procedural to an object-oriented style of programming. The gradual enrichment of sample classes with new options and their optimization, implemented through the application of newly introduced conceptions, plays a key role therein.