"Ss. Cyril and Methodius" University in Skopje
FACULTY OF COMPUTER
SCIENCE AND ENGINEERING

Object oriented programming

Exercises 8

Version 1.0, 11 April, 2017

Table of Contents

1. Polymorphism

1.1. Publications

Define a class for publication. Each publication is described with the year of publishing (integer) and the name of publishing house.

From this class derive (public) class for book. Each book additionally should have the number of pages.

Derive (protected) a second class named newspaper. Each newspaper has its number.

From the class for newspaper derive (private) a class for daily newspaper. Each daily newspaper has the day and month of publishing.

Review different levels of access to fields and functions!

Solution `oop_av81_en.cpp`

```cpp
#include <iostream>
#include <cstring>

using namespace std;
class Publication {
private:
    char name [100];
protected:
    int year;
    char* getName() {
        return name;
    }

public:
    int getYear() { return year;}

    void print () {
        cout << "Publication: " << name << " - " << year << endl;
    }

    Publication( char *name, int year ) {
        strcpy(this->name, name);
        this->year = year;
    }
};

// public inheritance

class Book: public Publication {
private:
    int number_of_pages;
public:
    Book(char *name, int year, int number_of_pages): Publication(name, year) {
        this->number_of_pages = number_of_pages;
    }
    void printBookYear() {
        cout << year; // access to protected resource year
    }
    void printBookName () {
        // access to getName(), name can be accessed because it's private
        cout << getName();
    }
    void numberOfPages() {
        cout << number_of_pages;
    }
};

// protected inheritance

class Newspaper: protected Publication {
private:
    int number;
public:
    Newspaper(char* name, int year, int number): Publication(name, year) {
        this->number = number;
    }
    void printYearNewspaper () {
        cout << getYear(); // access to public getYear(), that in this class has protected
access
    }
    void printNameNewspaper() {
        cout << getName(); // access to getName(), name that can not be accessed because
it is private
    }
    void printNumber() {
        cout << number;
    }
};

// private inheritance

class DailyNewspaper: private Newspaper {
private:
    int day;
    int month;
public:
    DailyNewspaper(char *name, int day, int month, int year,
                int number): Newspaper(name, year, number) {
        this->day = day;
        this->month = month;
    }
    using Newspaper::print; // function print from Publication becomes public for
DailyNewspaper
    using Newspaper::printNumber; // function printNumber from Publication becomes public
for DailyNewspaper
};

int main() {
    Publication p("Tabernakul", 1992);
    p.print(); // public - function
    Book *k = new Book("ProsvetnoDelo", 1900, 123);
    k->print(); //print is public in Book
    k->printBookYear(); // public - function
    // cout<<k->getName(); // error! protected - function
    Newspaper *s = new Newspaper("Tea", 2013 , 30);
    // s->print(); //error! protected - function
    // cout<<s->getYear(); // error! protected - function
    s->printNameNewspaper(); // public - function
    DailyNewspaper d("Vest", 2, 3, 2014, 25);
    d.print(); //public-function
    // d.printNameNewspaper(); // error! private - function
    // cout<<d.getName(); // error! private - function
}
```

1.2. Hotel reservation

Define a class for `HotelReservation` with number of days, number of persons and contact name. The price for the reservation for one person is 25 EUR per day.

In the class define function `price()` that returns the total price of the reservation. Also define function `price(int payment)` that returns the price with the given payment.

Derive a class `BBHotelReservation` for reservation only with breakfast. Price of the breakfast for one person is 5 EUR. Override the function `price(int payment)`.

Define a class `Hotel` with name of the hotel and the balance. Implement a function `int reserve(HotelReservation &hr, int payment)`. This function should make a payment for given hotel reservation. If the payment surpass the needed amount the function should return the change. The payment is made on the hotel balance.

What will happen if the argument is not reference?

Solution `oop_av82_en.cpp`

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

class HotelReservation {
protected:
    int days;
    int persons;
    char name[50];
    char surname[50];

public:
    HotelReservation(char *name, char *surname, int days, int persons) {
        strcpy(this->name, name);
        strcpy(this->surname, surname);
        this->days = days;
        this->persons = persons;
    }

    virtual int getPrice() {
        return days * persons * 25;
    }
    virtual int getPrice(int payment) {
        if (payment >= getPrice())
            return payment - getPrice();
        else {
            cout << "You should pay " << getPrice() << endl;
            return -1;
        }
    }
};


class BBHotelReservation: public HotelReservation {
public:
    BBHotelReservation(char *name, char *surname, int days, int
                                    persons) : HotelReservation(name, surname, days,
persons) {}

    //overriding getPrice(int payment)
    int getPrice(int payment) {
        int price = HotelReservation::getPrice() + persons * 5; // пристап до protected
податокот persons
        if (payment >= price)
            return payment - price;
        else {
            cout << "You should pay: " << price << endl;
            return -1;
        }
    }
};

class Hotel {
private:
    char name[50];
    int balance;
public:
    Hotel(char *name) {
        strcpy(this->name, name);
        balance = 0;
    }
    // reference of the base class that can reference objects from the derived classes
    int payForReservation(HotelReservation &hr, int payment) {
        int change = hr.getPrice(payment); // polymorphism
        // what definition of getPrice is going to be called?
        // important: getPrice() is virtual function
        if (change != -1)
            balance += payment - change;
        return change;
    }
};

int main() {
    Hotel h("Bristol");
    HotelReservation *hr1 = new HotelReservation("Petko", "Petkovski", 5, 5);
    int price = h.payForReservation(*hr1, 1000);
    if (price != -1)
        cout << "Change : " << price << endl;
    BBHotelReservation *hr2 =
        new BBHotelReservation("Risto", "Ristovski", 5, 5);
    price = h.payForReservation(*hr2, 1000);
    if (price != -1)
        cout << "Change : " << price << endl;
    // pointer to the base class pointing to object of derived
    HotelReservation *hr3 = new BBHotelReservation("Ana", "Anovska", 4, 2);
    price = h.payForReservation(*hr3, 100);
    if (price != -1)
        cout << "Change : " << price << endl;
    BBHotelReservation hr4("Tome", "Tomovski", 5, 3);
    price = h.payForReservation(hr4, 1000);
    if (price != -1)
        cout << "Change : " << price << endl;
}
```

1.3. Geometric shapes

Define an abstract class for geometric shape with height and base of different geometric figure. Define the following functions in the class: - `print()` that prints the info for the shape - `volume()` returns the volume of the shape - `height()` returns the height of the shape.

From the class geometric shape derive class for cylinder, cone and cuboid. For a cylinder and cone store the radius of the base. For the cuboid store the sides `a` and `b` of the basis.

In the `main` function declare and initialize dynamically allocated array of pointers of the class for geometric shape. From this array:

1. Find the shape with maximum volume using the global function: `void maxVolume(Shape *array[], int n);` This function should print the info for the shape with maximum volume.
2. Find the number of shapes that don't have basis circle using the global function: `double getRadius(Shape *s);`. This function returns the radius of the basis (if the basis is circle), -1 otherwise.

Solution `oop_av83_en.cpp`

```cpp
#include<iostream>
#include<cmath>
using namespace std;
class Shape {
protected:
    double height;
public:
    Shape (int height = 0) {
        this->height = height;
    }
    virtual void print() { //virtual function
        cout << height;
    }
    virtual double getVolume() = 0; //pure virtual function
    double getHeight() const {
        return height;
    }
};

class Cylinder: public Shape {
private:
    double radius;
public:
    //  constructor
    Cylinder(double radius, double height): Shape(height) {
        this->radius = radius;
    }
    // overriding function print()
    void print() {
        cout << "Cylinder with height ";
        Shape::print();
        cout << " and radius of tha basis" << radius << endl;
    }
    // overriding function getVolume()
    double getVolume() {
        return M_PI * radius * radius * getHeight();
    }
    double getRadius() {
        return radius;
    }
};

class Cone: public Shape {
private:
    double radius;
public:
    // constructor
    Cone(double radius, double height): Shape(height) {
        this->radius = radius;
    }
    // overriding function print()
    void print() {
        cout << "Cone with height ";
        Shape::print();
        cout << " and with radius of basis " << radius << endl;
    }
    // overriding function getVolume()
    double getVolume() {
        return M_PI * radius * radius * getHeight() / 3.0;
    }
    double getRadius() {
        return radius;
    }
};


class Cuboid: public Shape {
private:
    double a, b;
public:
    // constructor
    Cuboid(double a, double b, double height): Shape(height) {
        this->a = a;
        this->a = b;
    }
    // overriding function print()
    void print() {
        cout << "Cuboid with height ";
        Shape::print();
        cout << "and with basis " << this->a << " i " << this->b << endl;
    }
    // overriding function getVolume()
    double getVolume() {
        return a * b * getHeight();
    }
};
void maxVolume(Shape *shapes[], int n );
double getRadius (Shape *s);

int main() {
    Shape** shapes; // dynamically allocated array of pointers of Shape
    int n;
    cin >> n; // number of elements in the array
    shapes = new Shape*[n]; // allocate the pointer array
    for (int i = 0 ; i < n ; i++) {
        int r, a, b, h, type;
        cout << "Shape: 1-cylinder 2-cone 3-cuboid" << endl;
        cin >> type;
        if (type == 1) { // for Cylinder
            cin >> r >> h; shapes[i] = new Cylinder(r, h);
        } else if (type == 2) { // for Cone
            cin >> r >> h; shapes[i] = new Cone(r, h);
        } else if (type == 3) { // for Cuboid
            cin >> a >> b >> h; shapes[i] = new Cuboid(a, b, h);
        }
    }
    // 1.
    maxVolume(shapes, n);
    // 2.
    int counter = 0;
    for (int i = 0 ; i < n ; i++)
        if (getRadius(shapes[i]) == -1)
            counter++;
    cout << "Number of shapes with base circle is " << counter;
}
void maxVolume(Shape *shapes[], int n)
{
    int max = 0;
    int maxIndex = 0;
    for (int i = 0 ; i < n ; i++)
    {
        if (shapes[i]->getVolume() > max) {
            // call virtual function getVolume()
            max = shapes[i]->getVolume();
            maxIndex = i;
        }
    }
    cout << "Shape with max volume is:";
    shapes[maxIndex]->print(); // virtual call of function print()
}
double getRadius(Shape *g) {
    // runtime cast
    Cylinder* c = dynamic_cast<Cylinder *>(g);
    if (c != 0) { // if cast fails
        return c->getRadius();
    }
    // runtime cast
    Cone* k = dynamic_cast<Cone *>(g);
    if (k != 0 ) { // if cast fails
        return k->getRadius();
    }
    return -1; // if g is not a pointer to Cylinder or Cone return -1
}
```

2. Source code of the examples and problems

https://github.com/finki-mk/SP/

Source code ZIP