Računske vježbe 6

Programabilni uređaji i objektno orijentisano programiranje

Projektovati klase `Circle` (krug) i `Square` (kvadrat) koje su izvedene iz klase `Figure` (figura). Klasa figura sadrži težiste kao zajedničku katakteristiku za sve figure kao i metodu koja omogućava pomjeraj težista za zadatu vrijednost. Izvedene klase treba da imaju specifične metode za računanje obima i površine kao i očitavanje odgovarajućih podataka članova.

```
#include <iostream>
#include <cmath>

using namespace std;

class Point
{
private:
    double x; // koordinate
    double y;
public:
    Point(double x = 0, double y = 0) : x(x), y(y) {}
    double getX() const
    {
        return x;
    }
    double getY() const
    {
        return y;
    }
};

const Point ORIGIN; //koordinatni pocetak

class Figure
{
private:
    Point center; // teziste figure
public:
    Figure(Point center = ORIGIN) : center(center) {} // smjestamo figuru u
   koordinatni pocetak
    void shift(double x, double y) // pomjeraj tezista
    {
        center = Point(center.getX() + x, center.getY() + y);
    }
    void printCenter()
    {
        cout << " T=";
        cout << center.getX() << "," << center.getY();
    }
};

```

```
class Circle : public Figure
{
private:
    double radius;
public:
    Circle(double radius = 1, Point center = ORIGIN) : Figure(center), radius(radius)
    {}
    double perimeter() const
    {
        return 2 * radius * 3.14;
    }
    double area() const
    {
        return pow(radius, 2) * 3.14;
    }
    void print();
};

void Circle::print()
{
    cout << "U pitanju je krug: r=" << radius;
    printCenter();
    cout << " O=" << perimeter() << ", P=" << area() << endl;
};

class Square : public Figure
{
private:
    double side; // stranica kvadrata
public:
    Square(double side = 1.0, Point center = ORIGIN) : Figure(center), side(side) {}
    double perimeter() const
    {
        return 4 * side;
    }
    double area() const
    {
        return pow(side, 2);
    }
    void print();
};

void Square::print()
{
    cout << "U pitanju je kvadrat a=" << side;
    printCenter();
    cout << " O=" << perimeter() << " P=" << area() << endl;
};

int main()
{
    Circle circle(2, Point(3, 3));
    Square square(2.5, Point(1.3, 2));

    circle.print();
    square.print();

    circle.shift(1, 0.5);
    square.shift(0.5, 1);
```

```
100
101     circle.print();
102     square.print();
103 }
```

Klase koje opisuju podgrupu objekata s nekim dodatnim, specifičnim osobinama u odnosu na opštije grupe nazivaju se **izvedene klase**. Polazne klase, koje opisuju opštiju grupu objekata, nazivaju se **osnovne klase**. Izvođenje klase iz neke osnovne klase može biti javno, zaštićeno ili privatno, dodavanjem modifikatora `public`, `protected`, `private` ispred identifikatora osnovne klase. Kako način izvođenja utiče na vidljivost može se vidjeti u Tabeli 1. U odsustvu modifikatora podrazumijeva se privatno izvođenje. Izvedena klasa nasljeđuje sve članove svojih osnovnih klasa, ali se konstruktori, destruktor i metoda `operator=` kao i prijateljstva osnovne klase ne nasljeđuju. Jednostavan primjer izvođenja:

```
class A
{
protected: int i;
};
class B: public A
{
public: int j;
}
```

Mada se privatna polja mogu naslijediti, klasa koja ih naslijedi ih sadrži (zauzimaju memoriju), ali im ne može pristupiti. Da bi se izbjegla ova situacija, a da se ne bi narušila enkapsulacija, uvodi se modifikator `protected`. Pomoću ovog modifikatora činimo da za klasu B polje `A::i` ostane pristupačno, ali da se van klase B efektivno doživljava kao privatno polje. U praksi najveći značaj ima odnos *jeste* (objekat klase B jeste objekat klase A) između izvedenih i osnovnih klasa, pa se najčešće koristi javno izvođenje klasa, a zaštićeno i privatno samo u izuzetnim slučajevima. Izvedena klasa pored svojih članova posjeduje

način izvođenja	član osnovne klase		
	public	protected	private
public	public	protected	private
protected	protected	protected	private
private	private	private	private

Tabela 1: Stepeni zaštite članova osnovnih klasa u izvedenim klasama. Uočite da izvedena klasa može da zadrži vidljivost elemenata osnovne klase ili da vidljivost pogorša, ali ne može da popravlja vidljivost (recimo, sa privatne na javnu), jer bi na taj način ugrozili enkapsulaciju, odnosno potpunu kontrolu klase nad samom sobom.

i jedan bezimeni primjerak svoje osnovne klase, ali se njemu (jer nema ime) ne može direktno pristupati već samo njegovim članovima pojedinačno. Prilikom definisanja konstruktora za izvedenu klasu, u listi inicijalizatora prije tijela konstruktora, mogu da se navedu i inicijalizatori za osnovne klase. Međutim, u fazi inicijalizacije primjeraka klase ona polja koja su naslijeđena još ne postoje, pa ne mogu da se navedu odvojeni inicijalizatori za njih. Moguća je samo inicijalizacija naslijeđenog podobjekta tipa osnovne klase kao cjeline. U našem zadatku smo za klasu `Square` imali:

```
Square(double side = 1.0, Point center = ORIGIN) : Figure(center), side(side) {}
```