

Języki programowania obiektowego

Polimorfizm w pigułce



Odwołanie do obiektu przez wskaźnik

```
Kwadrat kwadr( 5 );
```

```
...
```

```
wypiszDane( &kwadr );
```

```
...
```

```
void wypiszDane( Kwadrat * k )
```

```
{
```

```
cout << "\nDługość boku : " << (*k).podajDłBoku();
```

```
cout << "\nPole kwadratu: " << (*k).obliczPole();
```

```
}
```

//ALBO ŁADNIEJ:

```
void wypiszDane( Kwadrat * k )
```

```
{
```

```
cout << "\nDługość boku : " << k->podajDłBoku();
```

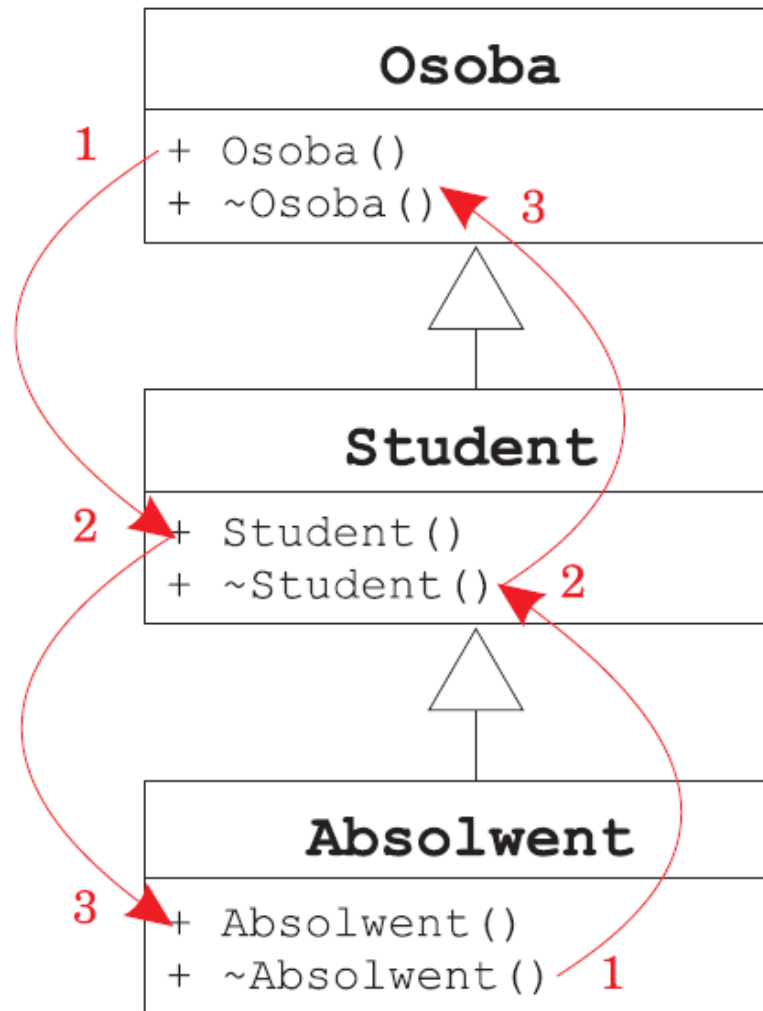
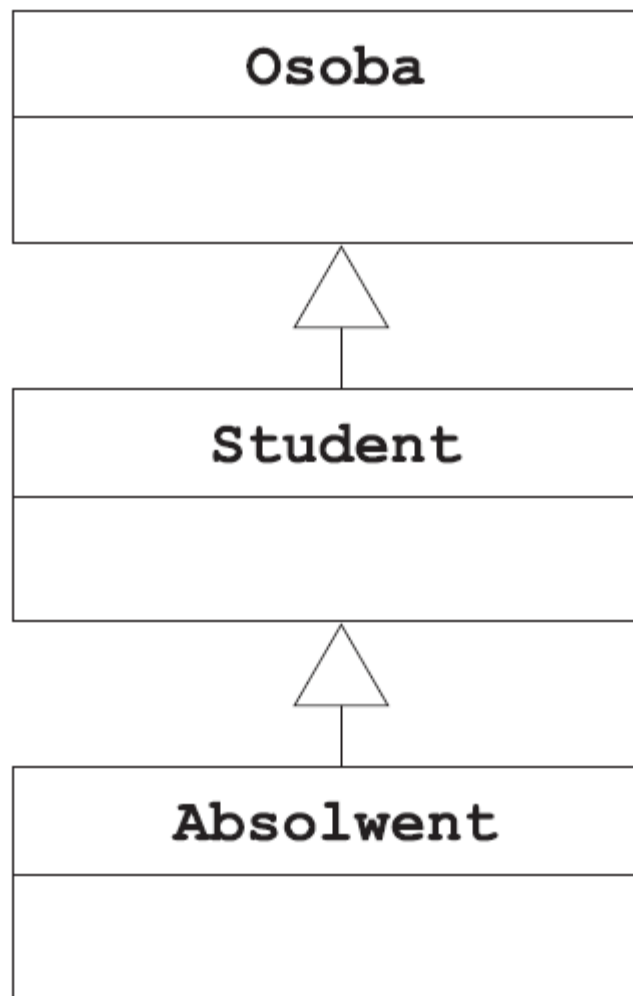
```
cout << "\nPole kwadratu: " << k->obliczPole();
```

```
}
```

Dynamiczny przydział pamięci

```
Kwadrat * kwadr;  
kwadr = new Kwadrat( 5 );  
if( kwadr != 0 )  
{  
cout << "\nDługosc boku : " << kwadr->podajDlBoku();  
cout << "\nPole kwadratu: " << kwadr->obliczPole();  
delete kwadr;  
}
```

Hierarchia klas – przypomnienie



Nowa hierarchia klas

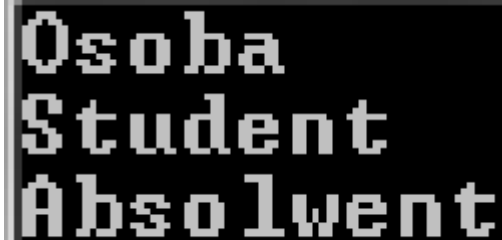
```
class Osoba
{
    public:
        Osoba() : nrDowodu(0) {}
        void piszKimJestes()
        {
            cout << "Osoba\n";
        }
        int nrDowodu;
};
```

```
class Student : public Osoba
{
    public:
        Student() : Osoba(),
            nrIndeksu(0) {}
        void piszKimJestes()
        {
            cout << "Student\n";
        }
        int nrIndeksu;
};
```

Nowa hierarchia klas c.d.

```
class Absolwent : public Student
{
    public:
        Absolwent() : Student(), nrDyplomu(0) {}
        void piszKimJestes()
        {
            cout << "Absolwent\n";
        }
        int nrDyplomu;
};

//gdzieś w main()
Osoba o;
Student s;
Absolwent a;
o.piszKimJestes();
s.piszKimJestes();
a.piszKimJestes();
```

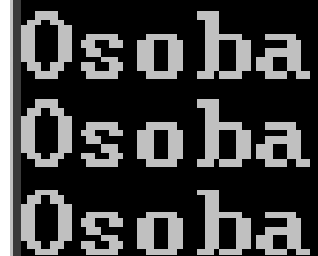


Osoba
Student
Absolwent

Polimorfizm a hierarchia klas

```
int main()
{
    Osoba o;
    Student s;
    Absolwent a;
    o.piszKimJestes();
    o = s;
    o.piszKimJestes();
    o = a;
    o.piszKimJestes();
}
```

```
int main()
{
    Osoba * wo;
    wo = &o;
    wo->piszKimJestes();
    wo = &s;
    wo->piszKimJestes();
    wo = &a;
    wo->piszKimJestes();
}
```



Osoba
Osoba
Osoba

Ratunek – dynamiczne wiązanie

```
class Osoba
{
    public:
        . . .
        virtual void piszKimJestes()
        {
            cout << "Osoba\n";
        }
        . . .
};
```

Osoba
Student
Absolwent

Lepszy przykład wykorzystania polimorfizmu

```
class Osoba
{
public:
    Osoba() : imie( "" ), nazwisko( "" ), nrDowodu( 0 ) {}
    Osoba( string i, string n, int nr ) : imie( i ),
        nazwisko( n ), nrDowodu( nr ) {}
    virtual void piszKimJestes() { cout << "\nOsoba\n"; }
    virtual void wypiszDane()
    {
        piszKimJestes();
        cout << imie << " " << nazwisko << endl;
        cout << "Nr dowodu: " << nrDowodu << endl;
    }
    int nrDowodu;
    string imie;
    string nazwisko;
};
```

Lepszy przykład wykorzystania polimorfizmu c.d.

```
class Student : public Osoba
{
public:
    Student() : Osoba(), nrIndeksu( 0 ) {}
    Student( string i, string n, int nr, int nrI )
    : Osoba( i, n, nr ), nrIndeksu( nrI ) {}
    void piszKimJestes() { cout << "\nStudent\n"; }
    void wypiszDane()
    {
        Osoba::wypiszDane();
        cout << "Nr indeksu: " << nrIndeksu << endl;
    }
    int nrIndeksu;
};
```

Lepszy przykład wykorzystania polimorfizmu c.d.

```
class Absolwent : public Student
{
public:
    Absolwent() : Student(), nrDyplomu( 0 ) {}
    Absolwent( string i, string n, int nr, int nrI, int
        nrD ): Student( i, n, nr, nrI ), nrDyplomu( nrD ){}
    void piszKimJestes() { cout << "\nAbsolwent\n"; }
    void wypiszDane()
    {
        Student::wypiszDane();
        cout << "Nr dyplomu: " << nrDyplomu << endl;
    }
    int nrDyplomu;
};
```

Wykorzystanie stworzonych klas

```
Osoba * wo = new Student("Jan", "Nowak", 2222, 3333);
```

```
. . .
```

```
wo->wypiszDane();
```

```
. . .
```

```
delete wo;
```

```
. . .
```

```
Osoba * wo = new Absolwent("Jan", "Nowak", 2222, 3333, 4444);
```

```
. . .
```

```
wo->wypiszDane();
```

```
. . .
```

```
delete wo;
```

```
Student
Jan Nowak
Nr dowodu: 2222
Nr indeksu: 3333
```

```
Absolwent
Jan Nowak
Nr dowodu: 2222
Nr indeksu: 3333
Nr dyplomu: 4444
```

Metody czysto wirtualne

```
class KlasaAbstrakcyjna
{
public:
    virtual void MetodaDoNadpisania() = 0;
}
```

Jeżeli w klasie jest przynajmniej jedna metoda czysto wirtualna to nazywamy ją abstrakcyjną.

Wirtualny destruktork

```
class Figura
{
public:
    virtual obliczPole() = 0;
    virtual ~Figura();
};
```

Wirtualny destruktork często pozwala uniknąć problemów z wyciekami pamięci.

Ćwiczenia

1. Przekształć poprzednie klasy dotyczące figur geometrycznych tak, by dziedziczyły po klasie bazowej **Figura**. W ramach klasy **Figura** zdefiniuj czysto wirtualną metodę **obliczPole()**. Następnie zdefiniuj w programie tablicę 3 różnych figur geometrycznych (np. zawierającą obiekty Kwadrat, Prostokąt, Sześciąt) oraz wypisz na ekranie sumę ich pól powierzchni (stosując odpowiednią iterację na w/w tablicy).

Ćwiczenia

2. Zaimplementuj następujące klasy:
 1. Klasę abstrakcyjną **Funkcja** – posiadającą pole x reprezentujące liczbę rzeczywistą oraz funkcję `wartość()`.
 2. Klasę **FunkcjaLiniowa** – dziedziczącą po klasie **Funkcja**. Jej podprogram `wartość()` powinien zwracać wynik wyrażenia $ax+b$.
 3. Klasę **FunkcjaKwadratora** – dziedziczącą po klasie **Funkcja**. Jej podprogram `wartość()` powinien zwracać wynik wyrażenia $ax^2 + bx + c$.

Przetestuj działanie stworzonych klas odwołując się do nich przez wskaźnik `Funkcja * test;`

Ćwiczenia

3. Napisz klasę **Towar**, posiadającą pola `nazwa`, `cena`, `liczba` i metodę `opis()`, wypisującą na ekran stan obiektu (wartości wszystkich pól).

Następnie zaimplementuj klasę **Lozko**, dziedziczącą w klasy **Towar**, ale posiadającą dodatkowo pola `długość`, `szerokość` i `wysokość`.

Ostatecznie stwórz funkcję `wypiszNaEkran(Towar * t[])`, która wypisuje informacje o wszystkich przekazanych w parametrze obiektach.

Propozycja projektu

Zaprojektować i zrealizować w technologii obiektowej prostą grę komputerową, w ramach której użytkownik steruje przy użyciu klawiatury ruchomym obiektem, poruszającym się w labiryncie (gra klasy *Pacman*).

Obiekt powinien zbierać rozmieszczone w labiryncie obiekty–nagrody, unikając jednocześnie obiektów–pułapek (np. min). Gra kończy się po zebraniu wszystkich nagród lub po utraceniu przez obiekt ruchomy „życia” w wyniku wejścia w pułapki.

Grę należy rozbudować o ruchome obiekty–realizujące aktywny pościg za sterowanym przez użytkownika obiektem ruchomym.



Wymogii

Na ocenę 3.0:

- ▶ Obiekt porusza się w konsoli pod wpływem działań użytkownika (naciśnięcia przycisków strzałek na klawiaturze).
- ▶ Obiekt nie może wyjść poza obszar planszy.
- ▶ Zjedzenie każdego obiektu nagrody skutkuje dodaniem jednego punktu do liczby ogółem uzyskanych przez gracza. Gra kończy się po zjedzeniu wszystkich nagród na planszy, wyświetleniem napisu o wygranej.
- ▶ Gra na dole planszy wyświetla aktualny stan zdobytych przez użytkownika punktów.

Na ocenę 4.0:

- ▶ Obiekt ma 3 życia do wykorzystania – najechanie na pułapkę powoduje utratę jednego życia oraz restart pozycji wszystkich obiektów. Utrata wszystkich 3 żyć, skutkuje napisem o przegranej.
- ▶ Istnieją co najmniej 2 rodzaje nagród, których zjedzenie powoduje dopisanie różnej liczby punktów.
- ▶ Po przegranej bądź wygranej, użytkownik może podać swoje imię i jego dane oraz wynik zapisywane są do pliku. Gra umożliwia jej rozpoczęcie lub wyświetlenie listy osób z pięcioma najwyższymi wynikami.

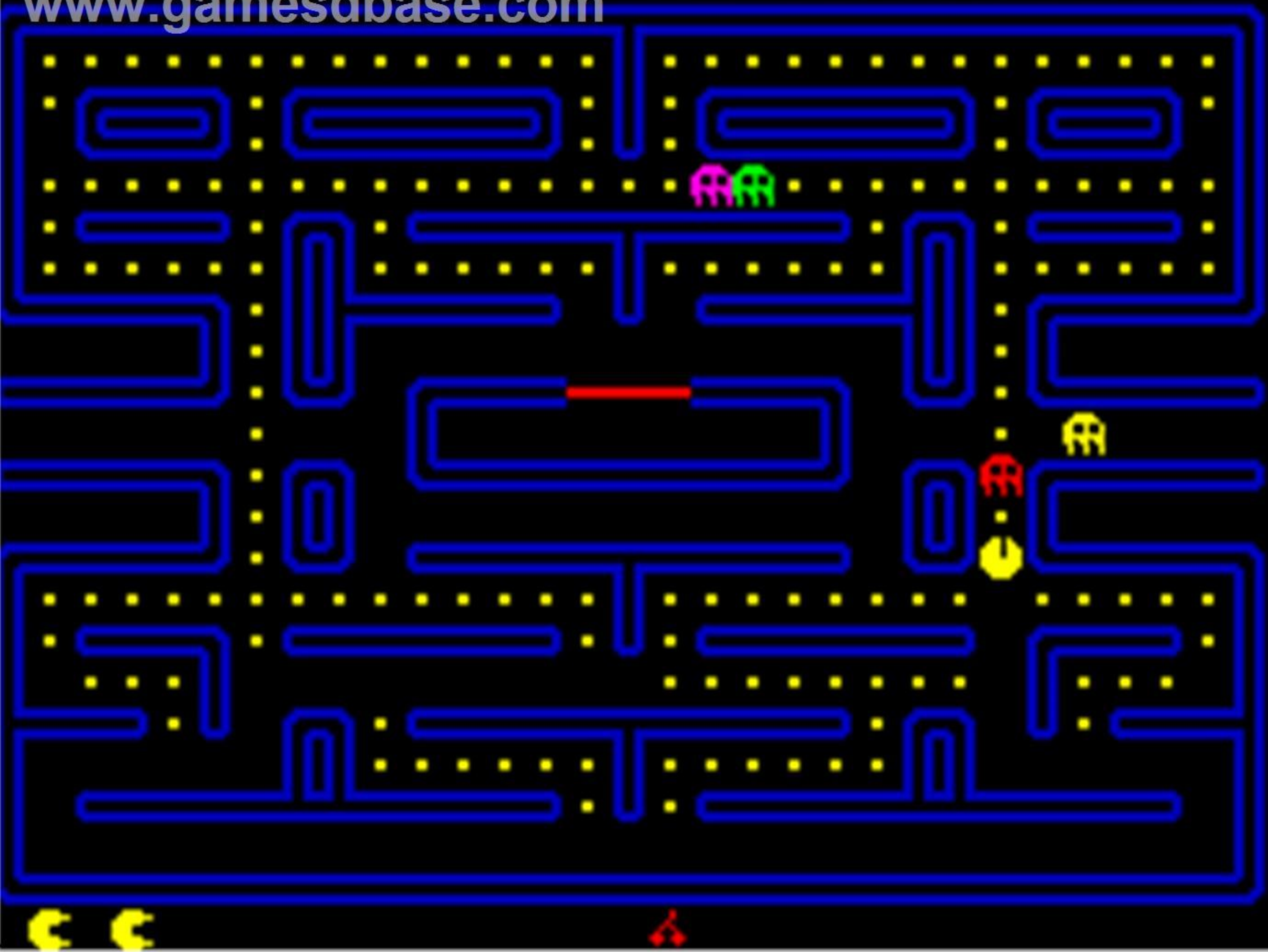
Na ocenę 5.0:

- ▶ Implementacja obiektów ścigających – poruszają się automatycznie po planszy, i gdy tylko w ich zasięgu znajdzie się obiekt poszukiwany, rozpoczynają za nim pościg. Zjedzenie przez obiekt ścigający powoduje utratę życia.

Program **musi** stosować zasady hermetyzacji, dziedziczenia i polimorfizmu. Tam gdzie to możliwe należy walidować dane od użytkownika

SCORE > 000580
www.gamesdbase.com

HI-SCORE > 005000



Jak to wszystko zrobić?

- ▶ Materiały z Wykładu dr Simińskiego oraz przykład programu **RunnerWithTrace**.

http://programowanie.siminskionline.pl/?page_id=7

- ▶ Bibliografia z C++.
- ▶ Pytania do prowadzącego.



Sterowanie konsolą pod Windows

```
#include <iostream>
#include <windows.h>

using namespace std;

HANDLE console = GetStdHandle(STD_OUTPUT_HANDLE);
COORD CursorPosition = {0, 0};

void gotoxy (int x, int y)
{
    CursorPosition.X = x; CursorPosition.Y = y;
    SetConsoleCursorPosition(console, CursorPosition);
}

int main()
{
    gotoxy(25,50);
    cout << "Hello world!" << endl;
    return 0;
}
```