Paradygmaty programowania obiektowego

Interfejsy w programowaniu obiektowym

dr inż. Radosław Idzikowski

Definicja

Interfejs - definicja abstrakcyjnego typu posiadającego jedynie operacje, a nie dane.

- w języku `C++` interfejs może być zdefiniowany jako klasa abstrakcyjna,
- w `Javie` czy `C#` stosuje się w tym celu specjalną deklarację ze słowem `interface`.

Budowa

Klasa abstrakcyjna jako interfejs:

- zawiera jedynie metody czysto wirtualne,
- może zawierać stałe,
- może zawierać metody zdefiniowane,
- metody mogą być publiczne oraz chronione,
- nie przechowuje żadnych danych!

Kiedy klasa definiuje wszystkie metody danego interfejsu, to mówimy, że implementuje interfejs.

Zastosowanie

Gdy kilka klas pochodnych wymaga użycia tej samej funkcji, ale z innymi jej implementacjami!

Klasa bazowa

```
class animal {
public:
    virtual void voice() const= 0;
    void test() const
    {
        std::cout << "elo!\n";
    }

};
```

Klasy pochodne

```
class dog : public animal {
public:
    void voice() const {
        std::cout << "woof!\n";
    }
};

class cat : public animal {
public:
    void voice() const {
        std::cout << "meow!\n";
    }
};
```

Wywołanie

```
int main()
{

    std::vector<animal*> animals;
    animals.push_back(new dog);
    animals.push_back(new cat);
    animals.push_back(new cat);
    animals.push_back(new dog);

    for (auto a : animals)
    {
        a->voice();
    }
}
```

Różnice

Klasa abstrakcyjna w języku C++, a interfejs w C#.

Klasy

```
interface Animal {

    void voice();
}

class Dog : Animal
{
    public void voice()
    {
        System.Console.WriteLine("woof!");
    }
}

class Cat : Animal
{
    public void voice()
    {
        System.Console.WriteLine("meow!");
    }
}
```

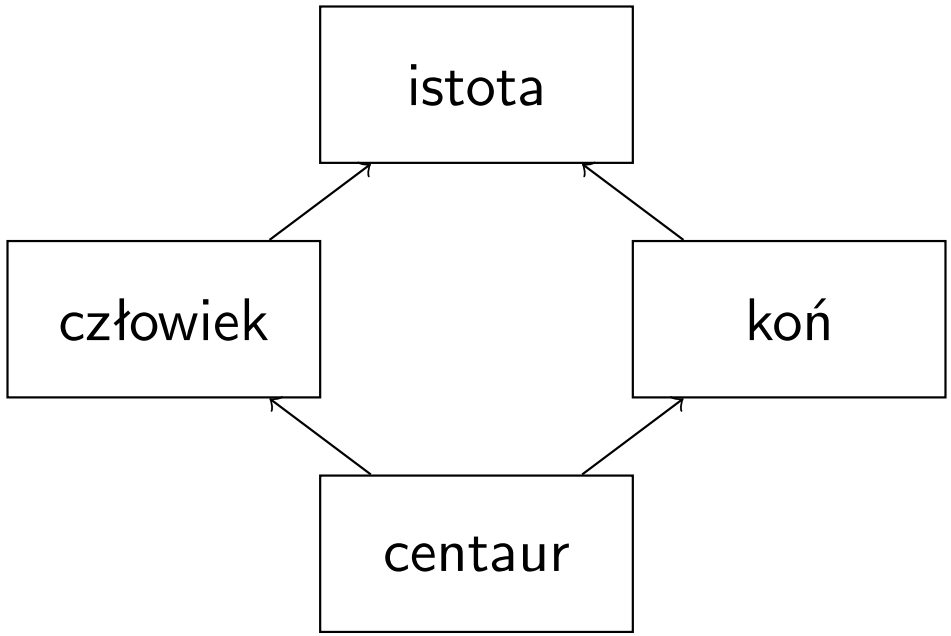
Wywołanie

```
static void Main(string[] args)
{
    List<Animal> list = new List<Animal>();
    list.Add(new Dog());
    list.Add(new Cat());
    foreach(Animal a in list)
    {
        a.voice();
    }

    System.Console.ReadLine();
}
```

Problem diamentu

Język C++ oferuje możliwość dziedziczenia wielokrotnego ze wszystkimi wadami i zaletami tego rozwiązania. W przypadku dziedziczenia po kilku klasach mających wspólnego przodka może pojawić się problem diamentu.



"base class is ambiguous!"

Klasa bazowa – istota

```
class Creature {
public:
    virtual void voice() const = 0;
};
```

Klasa pochodna – człowiek

```
class Human : public virtual Creature
{
public:
     void voice() const {
          std::cout << "Elo!\n";
     }
};
```

Klasa pochodna – koń

```
class Horse : public virtual Creature
{
public:
    void voice() const {
        std::cout << "Rrrrr!\n";
    }
};
```

Jeszcze jeden poziom – centaur

```
class Centaur : public Human, public Horse
{
    void voice() const {
        std::cout << "aaaa!\n";
    }
};
```

JWywołanie

```
int main()
{
    std::vector<Creature*> Creatures;
    Creatures.push_back(new Human());
    Creatures.push_back(new Horse());
    Creatures.push_back(new Centaur());
    for (auto a : Creatures)
    {
        a->voice();
    }
}
```