Dziedziczenie

C++

Dziedziczenie

Mechanizm współdzielenia funkcjonalności między klasami. Klasa może dziedziczyć po innej klasie, co oznacza, że oprócz swoich własnych atrybutów oraz zachowań, uzyskuje także te pochodzące z klasy dziedziczonej. Klasa dziedzicząca jest często nazywana klasą pochodną lub potomną (w j. angielskim: subclass lub derived class), zaś klasa, z której następuje dziedziczenie — klasą bazową (w ang. superclass). Z jednej klasy bazowej można uzyskać dowolną liczbę klas pochodnych. Klasy pochodne posiadają obok swoich własnych metod i pól, również kompletny interfejs klasy bazowej.

Klasy bazowe i klasy pochodne

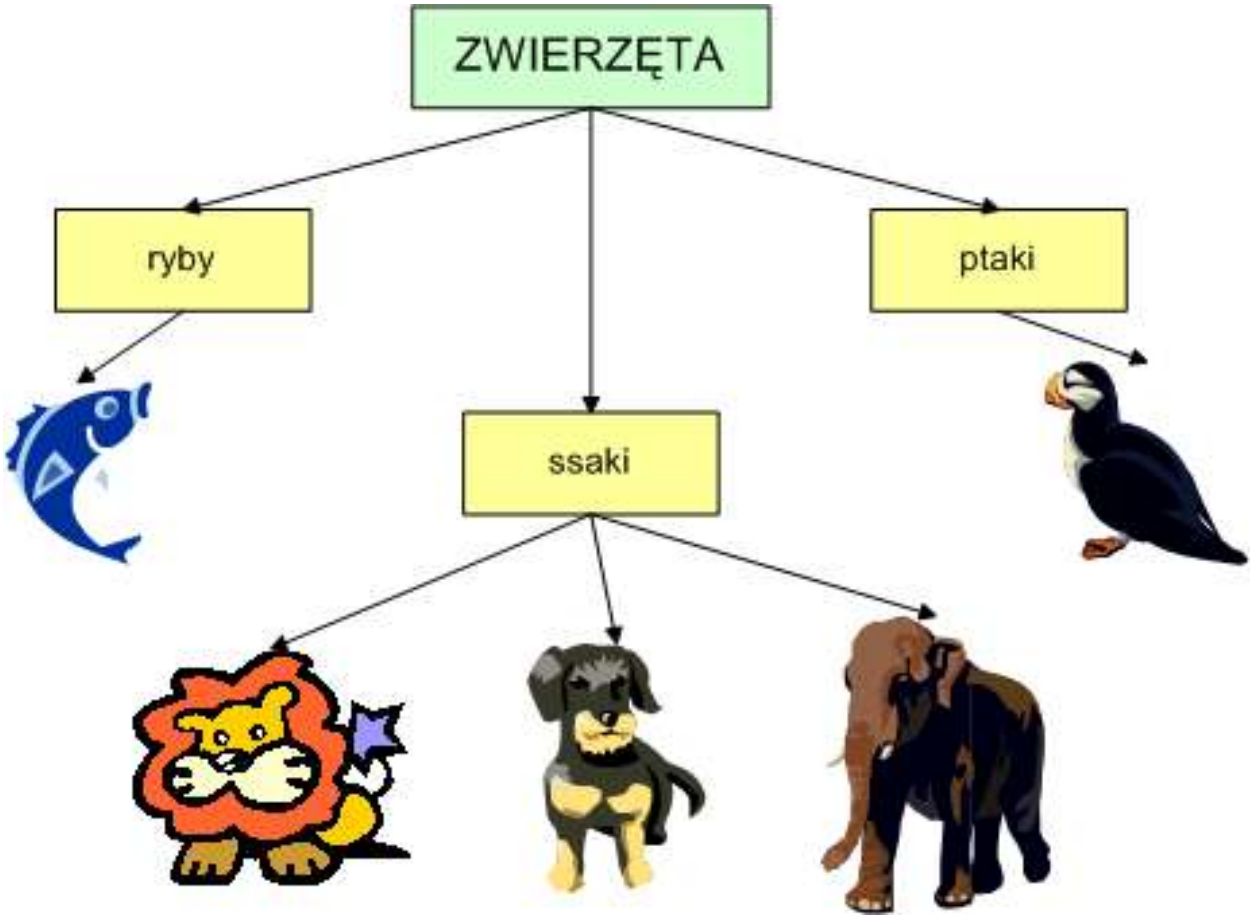
Zależności między klasami bazowymi i pochodnymi tworzą tzw. hierarchię klas. Klasy pochodne otrzymują wszystkie metody i atrybuty swoich klas bazowych oraz mogą dodawać nowe. Dopuszczalne jest także nadpisywanie istniejących metod, przy czym poszczególne języki programowania mogą żądać spełnienia dodatkowych warunków, np. pozostawienia niezmienionej listy argumentów wejściowych i typu wyniku.

ZYSK ze stosowania mechanizmu dziedziczenia

Ponowne wykorzystywanie raz napisanego kodu oraz rozszerzanie i dostosywanie go do własnych potrzeb.

Dziedziczenie – geneza powstania

Człowiek jest taką dziwną istotą, która bardzo lubi posiadać uporządkowany i usystematyzowany obraz świata. Wprowadzanie porządku i pewnej hierarchii co do postrzeganych zjawisk i przedmiotów jest dla nas niemal naturalną potrzebą.



Dziedziczenie (ang. *inheritance*) to tworzenie nowej klasy na podstawie jednej lub kilku istniejących wcześniej klas bazowych.

Wszystkie klasy, które powstają w ten sposób (nazywamy je pochodnymi), posiadają pewne elementy wspólne. Części te są dziedziczone z klas bazowych, gdyż tam właśnie zostały zdefiniowane.

Ich zbiór może jednak zostać poszerzony o pola i metody specyficzne dla klas pochodnych. Będą one wtedy współistnieć z "dorobkiem" pochodzącym od klas bazowych, ale mogą oferować dodatkową funkcjonalność.

Tak w teorii wygląda system dziedziczenia w programowaniu obiektowym. Najlepiej będzie, jeżeli teraz przyjrzymy się, jak w praktyce może wyglądać jego zastosowanie.

Przykład

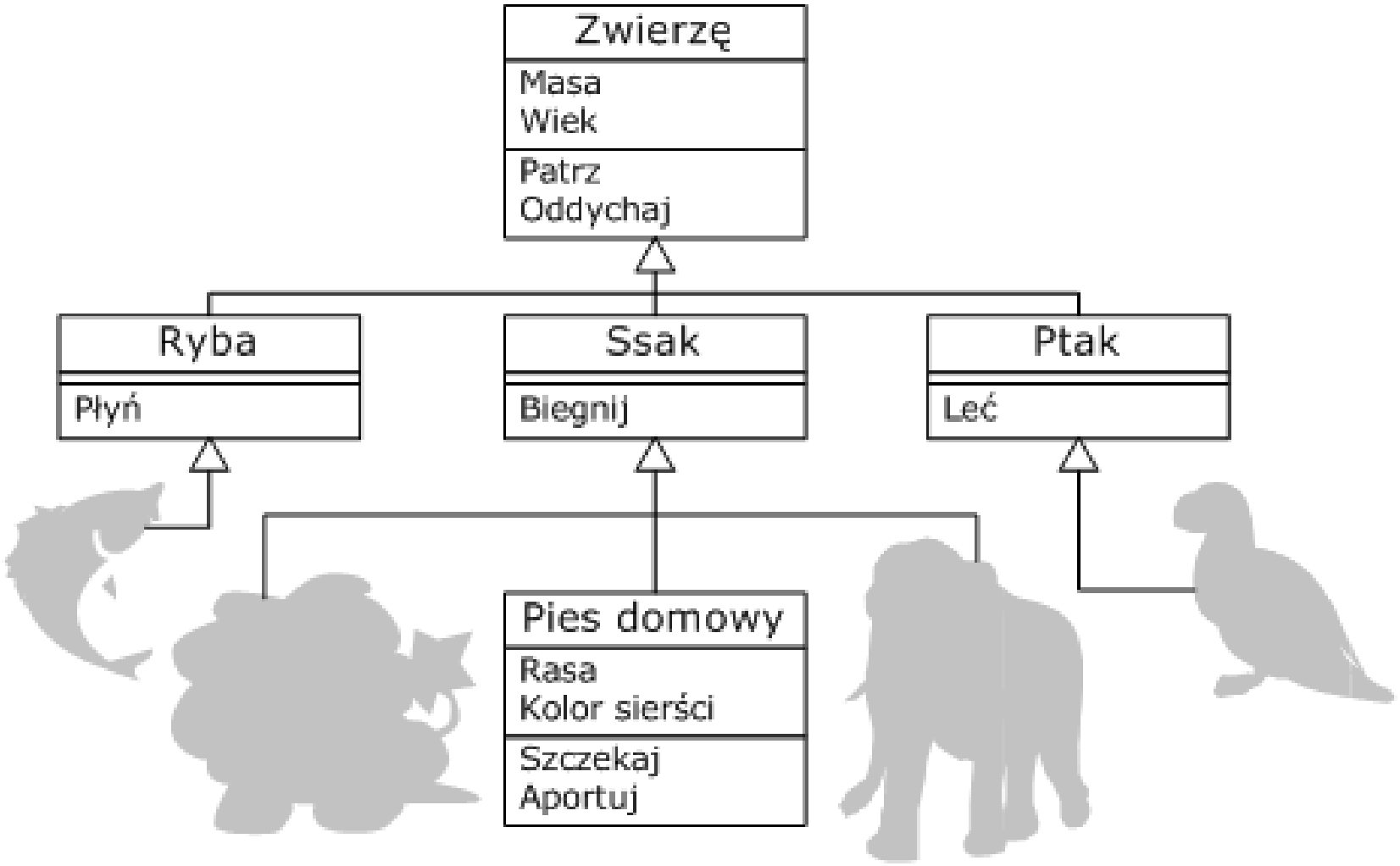
Wszystkie klasy, które powstają w ten sposób (nazywamy je pochodnymi **), posiadają pewne elementy wspólne. Części te są** dziedziczone **z klas bazowych, gdyż tam właśnie zostały zdefiniowane.**

Ich zbiór może jednak zostać poszerzony **o pola i metody specyficzne dla klas pochodnych. Będą one wtedy współistnieć z "dorobkiem" pochodzącym od klas bazowych, ale mogą oferować dodatkową funkcjonalność.**

Tak w teorii wygląda system dziedziczenia w programowaniu obiektowym. Najlepiej będzie, jeżeli teraz przyjrzymy się, jak w praktyce może wyglądać jego zastosowanie.

Pojęcie klas bazowych i klas pochodnych jest zatem względne : dana klasa może wprawdzie pochodzić od innych, ale jednocześnie być bazą dla kolejnych klas. W ten sposób ustala się wielopoziomowa hierarchia, podobna zwykle do drzewka.

Przykład Cd.


Zwierzę
Masa
Wiek
Patrz
Oddychaj
Ryba
Płyń
Ssak
Biegnij
Ptak
Leć
Pies domowy
Rasa
Kolor sierści
Szczekaj
Aportuj

Definicja klasy bazowej i specyfikator protected

Jak pamiętamy, definicja klasy składa się przede wszystkim z listy deklaracji jej pól oraz metod, podzielonych na kilka części wedle specyfikatorów praw dostępu. Najczęściej każdy z tych specyfikatorów występuje co najwyżej w jednym egzemplarzu, przez co składnia definicji klasy wygląda następująco:

```
class nazwa_klasy
{
[ private : ]
[ deklaracje_prywatne ]
[ protected : ]
[ deklaracje_chronione ]
[ public : ]
[ deklaracje_publiczne ]
};
```

Nieprzypadkowo pojawił się tu nowy specyfikator, protected . Jego wprowadzenie związane jest ściśle z pojęciem dziedziczenia. Pojęcie to wpływa zresztą na dwa pozostałe rodzaje praw dostępu do składowych klasy.

Specyfikatory Cd.

- **private : poprzedza deklaracje składowych, które mają być dostępne jedynie dla metod definiowanej klasy. Oznacza to, iż nie można się do nich dostać, używając obiektu lub wskaźnika na niego oraz operatorów wyłuskania . lub -> . Ta wyłączność znaczy również, że prywatne składowe nie są dziedziczone i nie ma do nich dostępu w klasach pochodnych, gdyż nie wchodzą w ich skład.**

- **protected ("chronione") także nie pozwala, by użytkownicy obiektów naszej klasy "grzebali" w opatrzonych nimi polach i metodach. Jak sama nazwa wskazuje, są one chronione przed takim dostępem z zewnątrz. Jednak w przeciwieństwie do deklaracji private , składowe zaznaczone przez protected są dziedziczone i występują w klasach pochodnych, będąc dostępnymi dla ich własnych metod.**

- **public jest najbardziej liberalnym specyfikatorem. Nie tylko pozwala na odziedziczanie swych składowych, ale także na udostępnianie ich szerokiej rzeszy obiektów poprzez operatory wyłuskania.**

Definicja klasy pochodnej

Dopiero posiadając zdefiniowaną klasę bazową możemy przystąpić do określania dziedziczącej z niej klasy pochodnej. Jest to konieczne, bo w przeciwnym wypadku kazalibyśmy kompilatorowi korzystać z czegoś, o czym nie miałby wystarczających informacji.

Składnię definicji klasy pochodnej możemy poglądowo przedstawić w ten sposób:

class *nazwa_klasy* **[** : **[** *specyfikatory* **]**
[*nazwa_klasy_bazowej* **] [** , ... **]]**
{
deklaracje_składowych
};

Specyfikatory , które opcjonalnie możemy umieścić przed każdą nazwą_klasy_bazowej . Wpływają one na proces dziedziczenia, a dokładniej na prawa dostępu , na jakich klasa pochodna otrzymuje składowe klasy bazowej.

Dziedziczenie pojedyncze

Najprostszą i jednocześnie najczęściej występującą w dziedziczeniu sytuacją jest ta, w której mamy do czynienia tylko z jedną klasa bazową **. Wszystkie dotychczas pokazane przykłady reprezentowały to zagadnienie; nazywamy je** dziedziczeniem pojedynczym **lub** jednokrotnym **(ang. *single inheritance*).**

Przykład - KOD

```
class CAnimal // Zwierzę
{
        protected :
          // pola klasy
          float m_fMasa;
          unsigned m_uWiek;
        public :
          // konstruktor
          CAnimal() { m_uWiek = 0 ; }
          //// metody
          void Patrz();
          void Oddychaj();
          // metody dostępowe do pól
          float Masa() const { return m_fMasa; }
          void Masa( float fMasa) { m_fMasa = fMasa; }
          unsigned Wiek() const { return m_uWiek; }
};

class CFish : public CAnimal // Ryba
{
        public :
          void Plyn();
};
```

```
class CMammal : public CAnimal // Ssak
{
        public :
          void Biegnij();
};
class CBird : public CAnimal // Ptak
{
        public :
          void Lec();
};

class CHomeDog : public CMammal // Pies domowy
{
        protected :
          // nowe pola
          RACE m_Rasa;
          COLOR m_KolorSiersci;
        public :
          // metody
          void Aportuj();
          void Szczekaj();
          // metody dostępowe do pól
          RACE Rasa() const { return m_Rasa; }
          COLOR KolorSiersci() const { return m_KolorSiersci; }
};
```