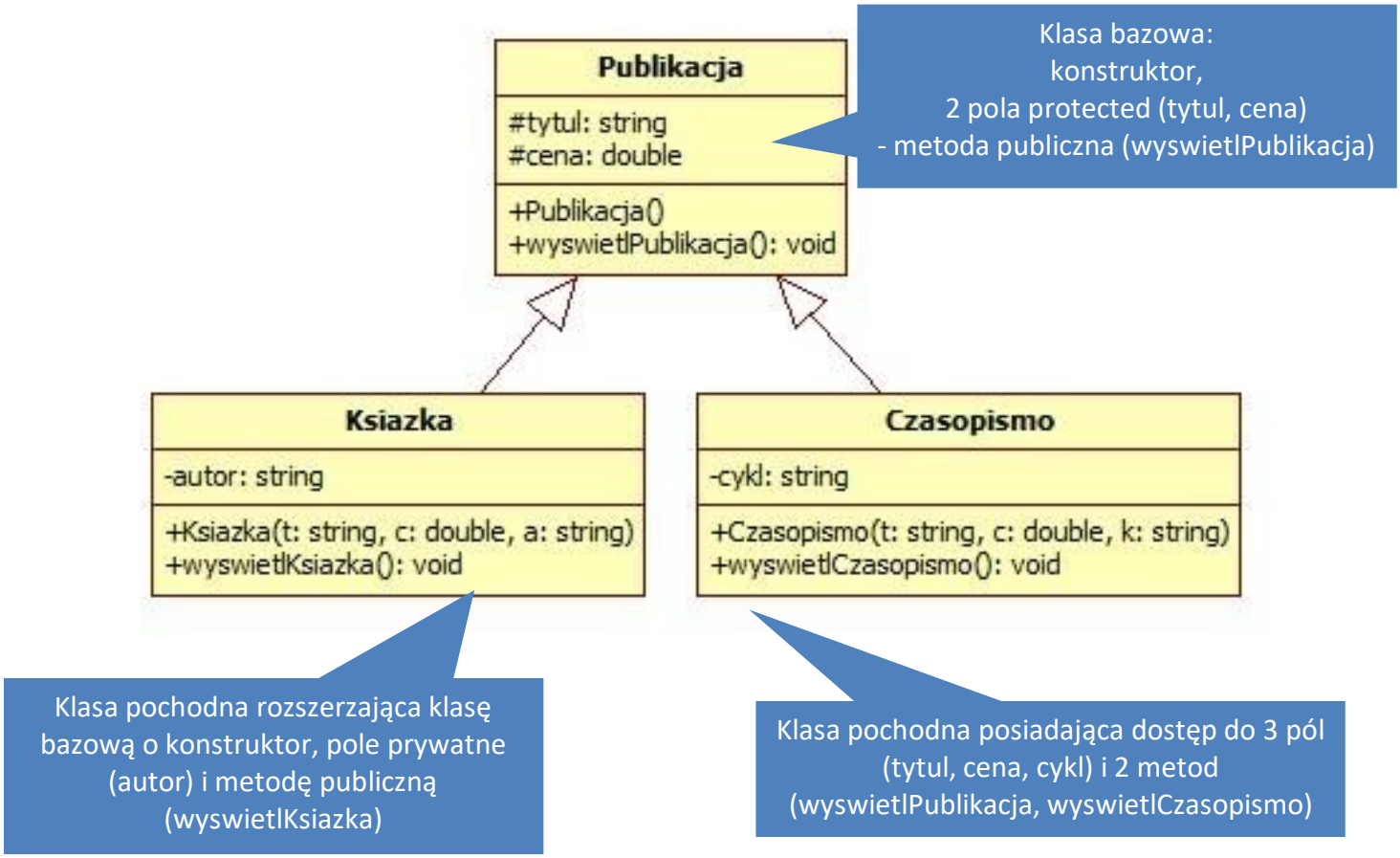
Temat: Dziedziczenie w języku C++

Dziedziczenie:

- **Pozwala stworzyć klasę na podstawie istniejącej już klasy.**
- **Klasą bazową** (podstawową, nadklasą) nazywamy klasę, z której dziedziczą inne klasy.
- **Klasa pochodną** (potomną, podklasą) nazywamy nową klasę, która dziedziczy strukturę innej klasy (jej pola i metody).

Przykładowy diagram klas:



Klasa bazowa Publikacja:

```
1   #include <iostream>
2   #include <string>
3   using namespace std;
4
5   class Publikacja {
6   protected:
7       string tytul;
8       double cena;
9   public:
10      Publikacja() {
11          tytul="Tytul publikacji";
12          cena=100;
13      }
14      void wyswietlPublikacja() {
15          cout<<"(Publikacja) tytul: "<<tytul<<endl;
16          cout<<"(Publikacja) cena: "<<cena<<endl;
17      }
18  };
19
```

Definicja i implementacja klasy bazowej **Publikacja**

Klasa potomna Ksiazka:

```
20  class Ksiazka : public Publikacja {
21  private:
22      string autor;
23  public:
24      Ksiazka(string t, double c, string a) {
25          tytul=t;
26          cena=c;
27          autor=a;
28      }
29      void wyswietlKsiazka() {
30          cout<<"(Ksiazka) tytul: "<<tytul<<endl;
31          cout<<"(Ksiazka) cena: "<<cena<<endl;
32          cout<<"(Ksiazka) autor: "<<autor<<endl;
33      }
34  };
35
```

Klasa **Ksiazka** dziedziczy po klasie bazowej **Publikacja**. Typ dziedziczenia: publiczny

Klasa potomna Czasopismo:

```
36  class Czasopismo : public Publikacja {
37  private:
38      string cykl;
39  public:
40      Czasopismo(string t, double c, string k) {
41          tytul=t;
42          cena=c;
43          cykl=k;
44      }
45      void wyswietlCzasopismo(){
46          cout<<"(Czasopismo) tytul: "<<tytul<<endl;
47          cout<<"(Czasopismo) cena: "<<cena<<endl;
48          cout<<"(Czasopismo) cykl: "<<cykl<<endl;
49      }
50  };
51
```

Program testujący:

```
52  int main() {
53      Publikacja p;
54      p.wyswietlPublikacja();
55
56      Czasopismo cz("Tytul czasopisma",200,"Cykl wydawniczy");
57      cz.wyswietlCzasopismo();
58      cz.wyswietlPublikacja();
59
60      Ksiazka k("Tytul ksiazki",300,"Jan Kowalski");
61      k.wyswietlKsiazka();
62      k.wyswietlPublikacja();
63
64      return 0;
65  }
```

```
(Publikacja) tytul: Tytul publikacji
(Publikacja) cena: 100
(Czasopismo) tytul: Tytul czasopisma
(Czasopismo) cena: 200
(Czasopismo) cykl: Cykl wydawniczy
(Publikacja) tytul: Tytul czasopisma
(Publikacja) cena: 200
(Ksiazka) tytul: Tytul ksiazki
(Ksiazka) cena: 300
(Ksiazka) autor: Jan Kowalski
(Publikacja) tytul: Tytul ksiazki
(Publikacja) cena: 300
```

Dziedziczenie wielokrotne:

- Dziedziczenie wielokrotne (wielobazowe, mnogie) pozwala na tworzenie klasy potomnej na podstawie więcej niż jednej klasy bazowej.

```
1  #include <iostream>
2  #include <string>
3  #include <cmath>
4  using namespace std;
5
6  class Okrag {
7  protected:
8      double r;
9  public:
10     Okrag(double r) { this->r=r; }
11     double obliczPole() { return M_PI*r*r; }
12 };
13
14 class Stol {
15 protected:
16     int kolor;
17 public:
18     Stol(int k) { kolor=k; }
19     void przemalujStol(int k) { kolor=k; }
20 };
21
22 class OkraglyStol : public Okrag, public Stol {
23 protected:
24     string nazwa;
25 public:
26     OkraglyStol(double r, int k, string n): Okrag(r), Stol(k) {
27         nazwa=n;
28     }
29     void wypisz() {
30         cout<<"nazwa="<<nazwa<<endl;
31         cout<<"r="<<r<<", kolor="<<kolor<<endl;
32     }
33 };
34
35 int main()
36 {
37     OkraglyStol os(75,13,"model deluxe");
38     os.wypisz();
39     cout<<"pole="<<os.obliczPole()<<endl;
40     return 0;
41 }
```

Dwukrotne dziedziczenie

Uruchamiamy konstruktory bazowe z argumentami

```
nazwa=model deluxe
r=75, kolor=13
pole=17671.5
```

Dziedziczenie wirtualne:

- Dziedzicznie wirtualne jest specjalnym przypadkiem w dziedziczeniu wielokrotnym, które stosuje się, gdy z jakiegoś powodu jedna z klas staje się wielokrotnie przodkiem (bezpośrednio lub pośrednio) innej klasy.
- Dziedziczenie wirtualne stosujemy w przypadku, aby uniknąć wielokrotnego dziedziczenia takich samych składowych.

```
1  #include <iostream>
2  using namespace std;
3
4  class KlasaA {
5      int a[1000];
6  };
7
8  class PochA1 : public KlasaA {};
9  class PochA2 : public KlasaA {};
10 class KlasaAA : public PochA1, public PochA2 {};
11
12 class PochB1 : virtual public KlasaA {};
13 class PochB2 : virtual public KlasaA {};
14 class KlasaBB : public PochB1, public PochB2 {};
15
16 int main()
17 {
18     KlasaAA a;
19     KlasaBB b;
20     cout<<sizeof(a)<<endl;
21     cout<<sizeof(b)<<endl;
22
23     return 0;
24 }
```

KlasaAA odziedziczyła podwójne pole a.

KlasaBB odziedziczyła tylko raz pole a.

```
8000
4008
```

Odwołanie do składowych klas pochodnych:

```
1  #include <iostream>
2  using namespace std;
3
4  class KlasaA {
5  public:
6      int a;
7  };
8  class KlasaB : public KlasaA {
9  public:
10     int a;
11     void wypisz() {
12         cout<<"(KlasaA) a="<<KlasaA::a<<endl;
13         cout<<"(KlasaB) a="<<KlasaB::a<<endl;
14     }
15 };
16
17 int main()
18 {
19     KlasaB obiekt;
20     obiekt.KlasaA::a=1;
21     obiekt.KlasaB::a=2;
22     obiekt.wypisz();
23     obiekt.a=3;
24     obiekt.wypisz();
25     return 0;
26 }
```

Zauważ, że w KlasaA i w KlasaB istnieją różne pola i tej samej nazwie.

```
(KlasaA) a=1
(KlasaB) a=2
(KlasaA) a=1
(KlasaB) a=3
```

Metody wirtualne:

- Metody wirtualne definiujemy w klasie bazowej.
- Metody wirtualne redefiniujemy w klasie pochodnej.
- Przeciążenie metody wirtualnej nie jest konieczne we wszystkich klasach pochodnych.
- Jeśli w klasie pochodnej nie ma redefinicji klasy wirtualnej to zostanie wykonana metoda z najwyższej klasy poprzedniej w hierarchii klas.

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

class Figura {
public:
    virtual double pole() { return 0; }
    void wypisz() { cout<<"Pole="<<pole()<<endl; }
};

class Kwadrat : public Figura {
private:
    double a;
public:
    Kwadrat(double a) { this->a=a; }
    virtual double pole() { return a*a; }
};

int main()
{
    Kwadrat kw(8);
    kw.wypisz();
    return 0;
}
```

```
Pole=64
```

Metoda **wypisz()** z klasy bazowej wywołuje funkcję wirtualną z klasy potomnej

Klasy abstrakcyjne:

- Klasy abstrakcyjne nie zawierają składowych prywatnych.
- Klasy abstrakcyjne nie zawierają definicji funkcji wirtualnych (tylko ich nagłówki).
- Klasy abstrakcyjne muszą mieć przynajmniej jedną metodę czysto wirtualną - czyli metodę wirtualną, która nie ma ciała.
- Klasy abstrakcyjne tworzymy, aby po nich dziedziczyć.
- Nie tworzy się obiektów klas abstrakcyjnych.

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

class Figura {                  //klasa abstrakcyjna
public:
    virtual double pole() = 0; //metoda czysto wirtualna
    void wypisz() { cout<<"Pole="<<pole()<<endl; }
};

class Kwadrat : public Figura {
private:
    double a;
public:
    Kwadrat(double a) { this->a=a; }
    virtual double pole() { return a*a; }
};

int main()
{
    Kwadrat kw(8);
    kw.wypisz();
    return 0;
}
```

```
Pole=64
```

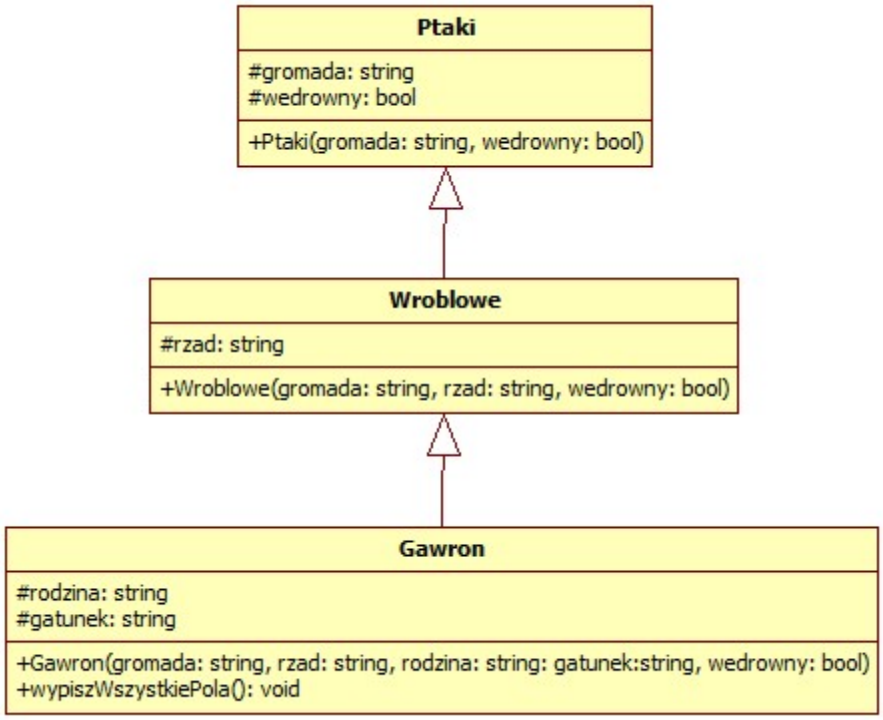
Wskaźnik na obiekt:

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

class Figura {                  //klasa abstrakcyjna
public:
    virtual double pole() = 0; //metoda czysto wirtualna
    void wypisz() { cout<<"Pole="<<pole()<<endl; }
};

class Kwadrat : public Figura {
private:
    double a;
public:
    Kwadrat(double a) { this->a=a; }
    virtual double pole() { return a*a; }
};

int main()
{
    Kwadrat *wsk = new Kwadrat(8); //wskaźnik na obiekt
    wsk->wypisz();
    delete wsk;                    //usunięcie obiektu
    return 0;
}
```

```
Pole=64
```

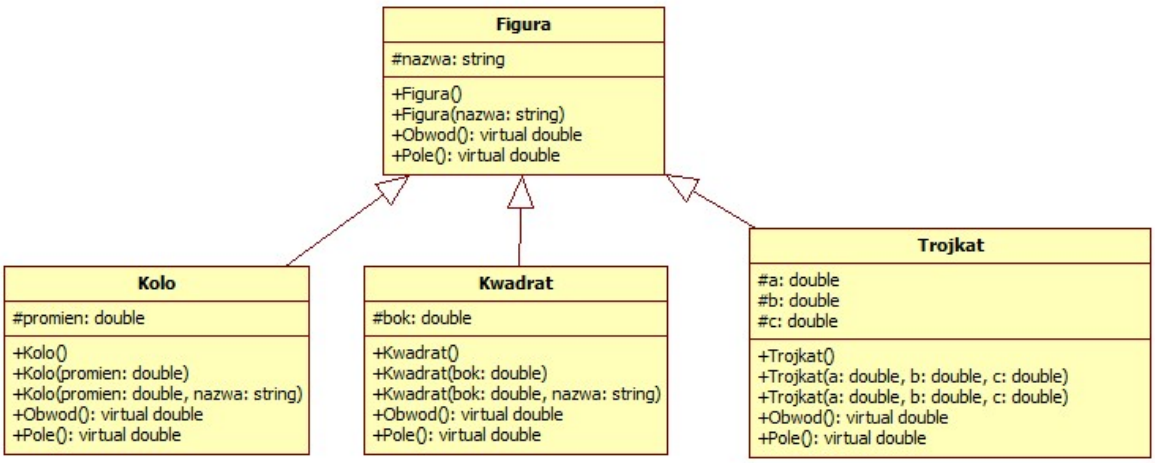
Zadania do samodzielnej realizacji:

1. (2p) Zaimplementuj klasy *Ptaki*, *Wroblowe*, *Gawron* zgodnie z podanym diagramem:



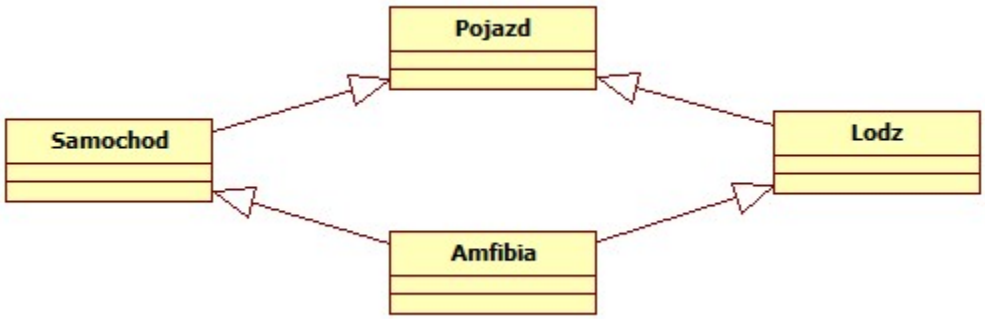
- Metoda *wypiszWszystkiePola()* z klasy *Gawron* powinna w sposób czytelny wypisywać wszystkie dostępne pola.
- Utwórz wskaźnik na obiekt klasy *Gawron* i przetestuj działanie metody *wypiszWszystkiePola().*

2. (3p) Zaimplementuj klasę abstrakcyjną *Figura* i klasy pochodne *Kolo*, *Kwadrat* i *Trojkat* oraz odpowiednie metody zgodnie z podanym diagramem:



- Konstruktory domyślnie powinny inicjować długości boków/promienia zerami zaś pole nazwa powinno zostać zainicjowane łańcuchami znaków odpowiednio: "Figura", "Kwadrat", "Kolo" i "Trojkat".
- Metody *Obwod()* i *Pole()* klasy *Figura* powinny być czysto wirtualne (nie powinny posiadać definicji).
- Stwórz metodę **wypisz()** w taki sposób, aby jej użycie:
  Kwadrat k(6);
  k.wypisz();
  Trojkat t(3,4,5);
  t.wypisz();
  Generowało następujący komunikat:
  Kwadrat o polu 36 i obwodzie 24.
  Trojkat o polu 6 i obwodzie 12.

3. (3p) Stwórz klasy *Pojazd*, *Samochod*, *Lodz*, *Amfibia* zgodnie z podanym diagramem:



Do każdej klasy dodaj zdefiniuj:

- Konstruktor bezargumentowy inicjujący wszystkie pola klasy wartościami domyślnymi,
- Konstruktor zawierający co najmniej jeden argument inicjujący pola klasy wartościami argumentów,
- Destruktor wypisujący na ekranie wartości wszystkich dostępnych pól,
- Co najmniej jedno pole protected własnego pomysłu,
- Co najmniej 2 metody publiczne własnego pomysłu.

Zadbaj o to, aby klasa *Amfibia* nie dziedziczyła dwukrotnie pól z klasy *Pojazd* (odpowiednie dziedziczenie powinno być wirtualne).
Utwórz wskaźniki na obiekty wszystkich klas i przetestuj działanie wszystkich utworzonych metod.
Na koniec usuń wszystkie utworzone obiekty.