

Programowanie w języku C++ (EAR1S03006)

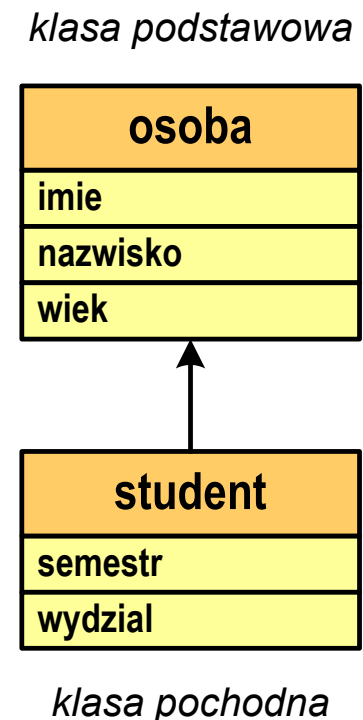
Politechnika Białostocka - Wydział Elektryczny
Automatyka i Robotyka, semestr III, studia stacjonarne I stopnia
Rok akademicki 2021/2022

Zajęcia nr 6 (17.11.2021)

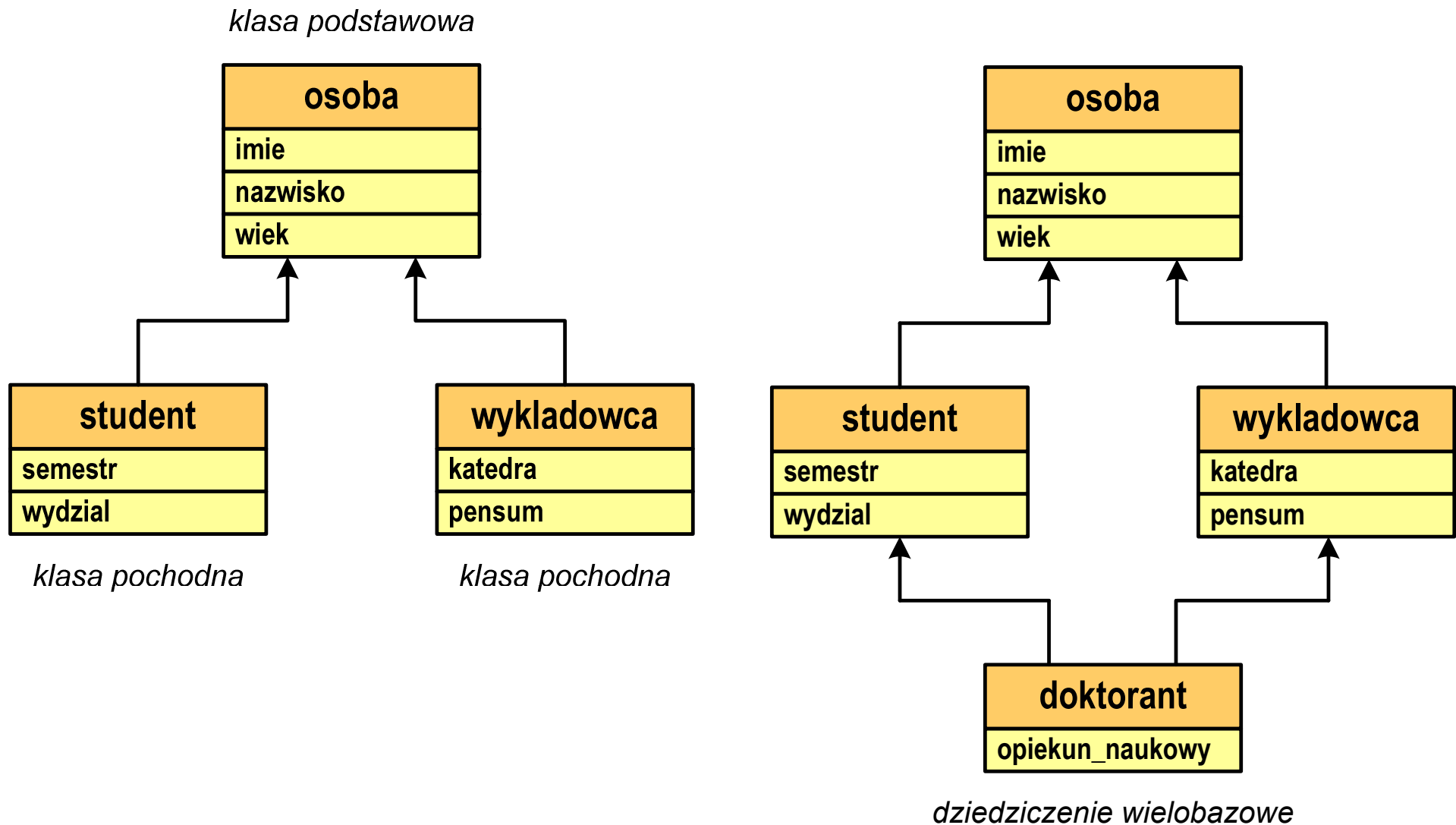
dr inż. Jarosław Forenc

Dziedziczenie

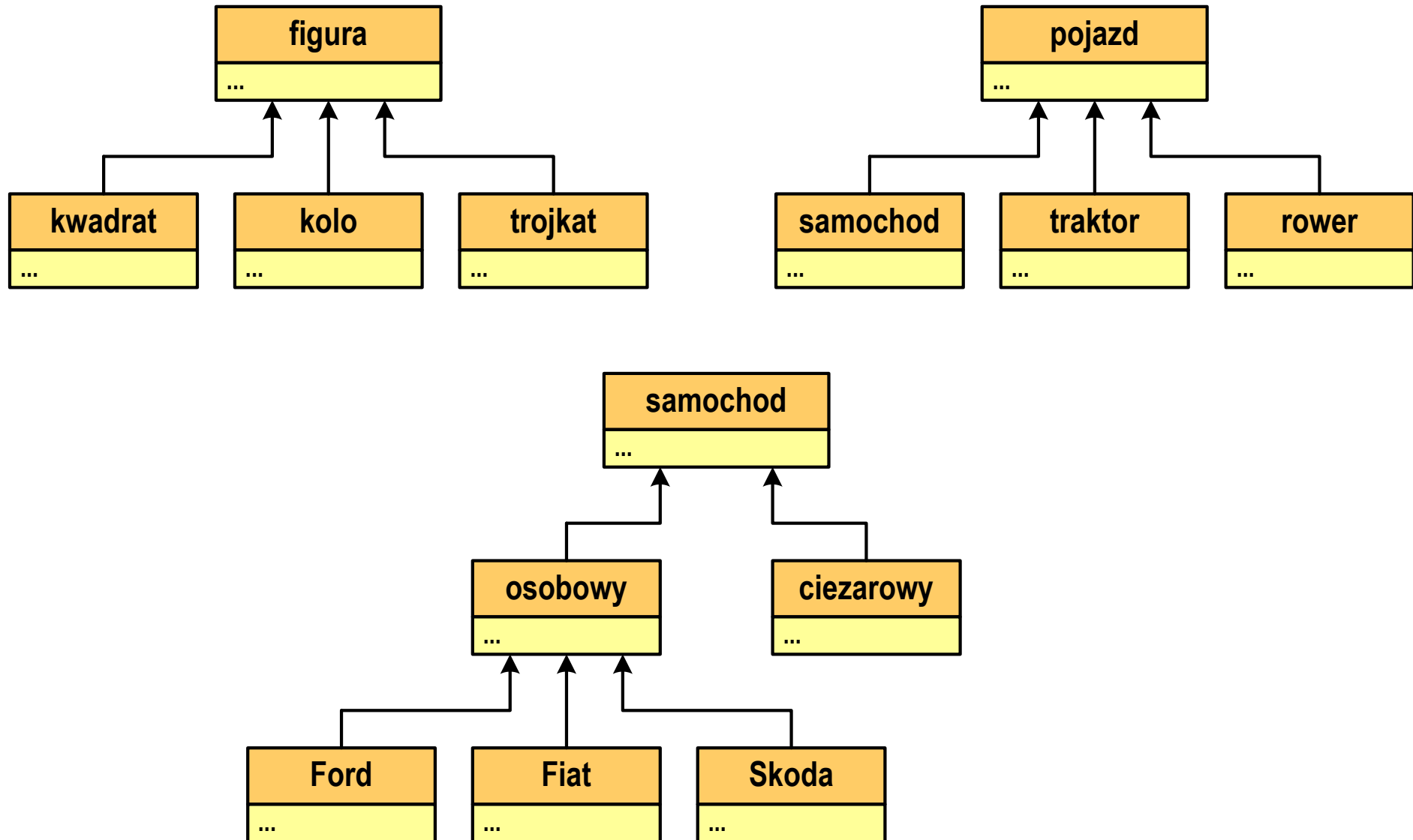
- ❑ **dziedziczenie** jest to technika pozwalająca na definiowanie nowej klasy przy wykorzystaniu klasy już istniejącej
- ❑ polega na przejmowaniu jednej klasy (**bazowej**, **podstawowej**) przez inną klasę (**pochodną**)
- ❑ przy dziedziczeniu, w skład obiektów klasy pochodnej automatycznie wchodzi pola klasy bazowej
- ❑ do obiektów klasy pochodnej możemy stosować operacje zdefiniowane przez funkcje składowe klasy bazowej



Dziedziczenie - przykłady



Dziedziczenie - przykłady



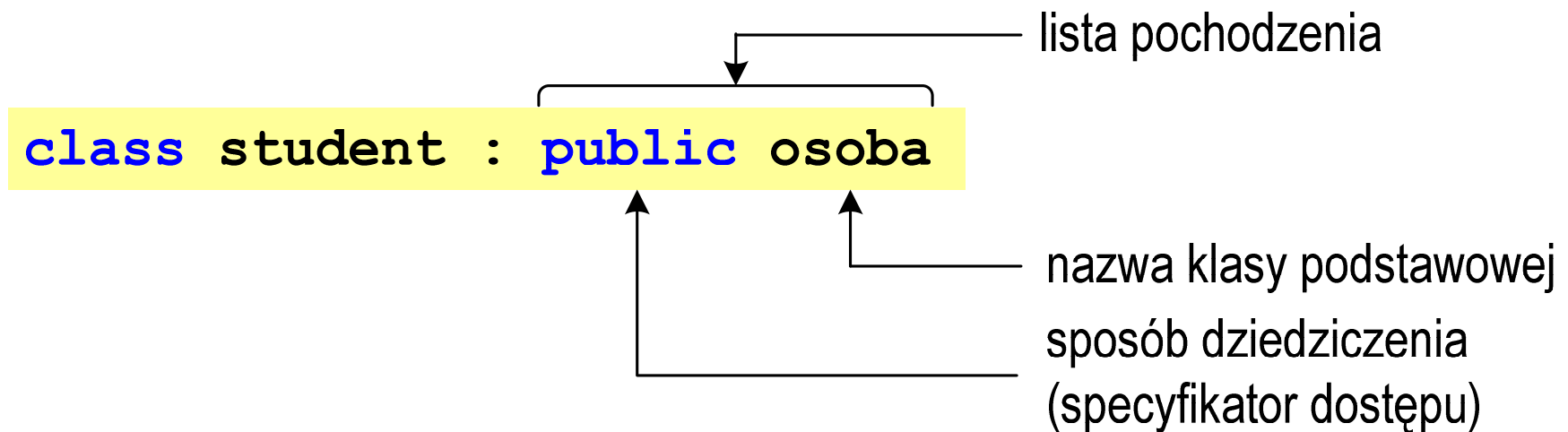
Dziedziczenie

- przykładowa klasa podstawowa i klasa pochodna

```
/* klasa podstawowa */  
  
class osoba  
{  
    char *imie;  
    char *nazwisko;  
    int  wiek;  
public:  
    osoba(char *i, char *n, int w);  
    ~osoba()  
    void drukuj();  
};
```

```
/* klasa pochodna */  
  
class student : public osoba  
{  
    char *wydzial;  
    int  semestr;  
public:  
    student(char *i, char *n,  
            int w, char *wy, int s);  
    ~student()  
    void drukuj();  
    void promocja();  
};
```

Dziedziczenie



- w klasie pochodnej można zdefiniować:
 - dodatkowe dane składowe
 - dodatkowe funkcje składowe
 - dane i funkcje o takich samych nazwach jak w klasie podstawowej (dane i funkcje z klasy podstawowej są zasłaniane)
- jeśli nie podamy sposobu dziedziczenia, to domyślnie będzie to **private**

Dziedziczenie - sposoby dziedziczenia

sposób dziedziczenia klasa podstawowa	private	protected	public
private	-	-	-
protected	private	protected	protected
public	private	protected	public

- podczas dziedziczenia nie są dziedziczone: konstruktor, destruktor i operator przypisania "="

Przykład:

```
student st1("Jan","Kos",20,"WE",2);  
st1.drukuj();  
st1.osoba::drukuj();
```

- deklaracja obiektu
- wywołanie funkcji z klasy **student**
- wywołanie funkcji z klasy **osoba**

- możliwe jest dziedziczenie wielokrotne, tzn. klasa pochodna może być klasą podstawową dla innej klasy

Przykład: dziedziczenie (1/4)

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

```
class osoba
{
    private:
        char *imie;
        char *nazwisko;
        int  wiek;
    public:
        osoba(char*, char*, int);
        ~osoba();
        void drukuj();
};
```

klasa podstawowa
(bazowa)

```
class student : public osoba
{
    private:
        char *wydzial;
        int  semestr;
    public:
        student(char*, char*, int,
                char*, int);
        ~student();
        void drukuj();
        void promocja();
};
```

klasa pochodna

Przykład: dziedziczenie (2/4)

```
osoba::osoba(char *i, char *n, int w)
{
    imie = new char[strlen(i)+1];
    nazwisko = new char[strlen(n)+1];
    strcpy(imie,i);
    strcpy(nazwisko,n);
    wiek = w;
}
```

konstruktor klasy osoba

```
osoba::~osoba()
{
    delete [] imie;
    delete [] nazwisko;
}
```

destruktor klasy osoba

```
void osoba::drukuj()
{
    cout << imie << " " << nazwisko;
    cout << " " << wiek << endl;
}
```

Przykład: dziedziczenie (3/4)

```
student::student(char *i, char *n, int w, char *wy, int s) : osoba(i, n, w)
{
    wydzial = new char[strlen(wy)+1];
    strcpy(wydzial, wy);
    semestr = s;
}

student::~~student()
{
    delete [] wydzial;
}

void student::drukuj()
{
    osoba::drukuj();
    cout << "Wydzial: " << wydzial;
    cout << " Semestr: " << semestr << endl;
}
```

konstruktor klasy student

destruktor klasy student

lista inicjalizacyjna konstruktora klasy student zawierająca wywołanie konstruktora klasy podstawowej (osoba)

Przykład: dziedziczenie (4/4)

```
void student::promocja()
{
    semestr++;
}

int main(void)
{
    student st1("Jan", "Kowalski", 20, "WE", 2);

    st1.drukuj();
    st1.promocja();
    st1.drukuj();
    st1.osoba::drukuj();
}
```

jako pierwszy zostanie wywołany konstruktor klasy podstawowej (osoba) a po nim konstruktor klasy pochodnej (student)

- kolejność wywołania **konstruktorów**:
 - konstruktor klasy podstawowej
 - konstruktor obiektów składowych „goszczących” w klasie pochodnej
 - konstruktor klasy pochodnej

Przykład: dziedziczenie (4/4)

```
void student::promocja()
{
    semestr++;
}

int main(void)
{
    student st1("Jan", "Kowalski", 20, "WE", 2);

    st1.drukuj();
    st1.promocja();
    st1.drukuj();
    st1.osoba::drukuj();
}
```

jako pierwszy zostanie wywołany konstruktor klasy podstawowej (osoba) a po nim konstruktor klasy pochodnej (student)

- kolejność wywołania **destruktorów** jest odwrotna w stosunku do konstruktorów - jako pierwszy jest wywoływany destruktor klasy **student**, a po nim destruktor klasy **osoba**

Przykład: dziedziczenie (4/4)

```
void student::promocja()
{
    semestr++;
}

int main(void)
{
    student st1("Jan", "Kowalski", 20, "WE", 2);

    st1.drukuj();
    st1.promocja();
    st1.drukuj();
    st1.osoba::drukuj();
}
```

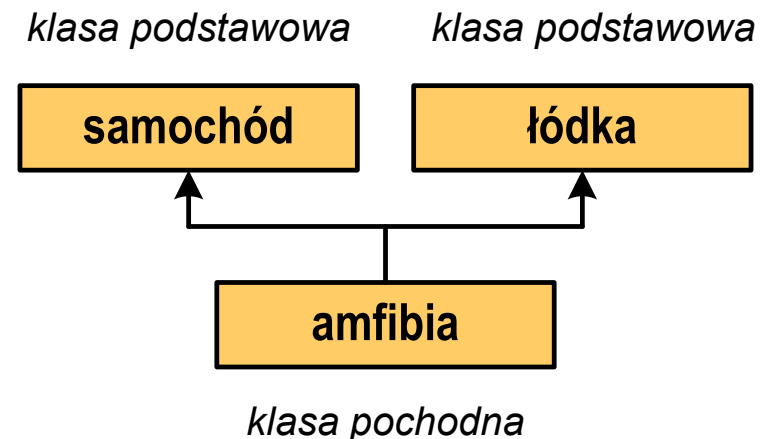
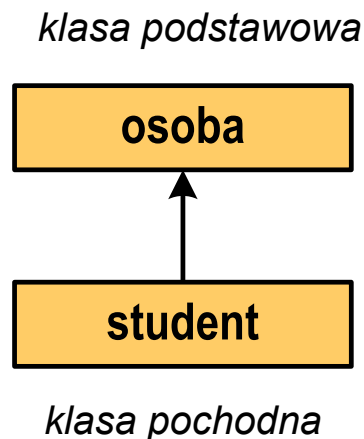
jako pi
konst
(osoba)

```
Jan Kowalski 20
Wydzial: WE Semestr: 2
Jan Kowalski 20
Wydzial: WE Semestr: 3
Jan Kowalski 20
```

- kolejność wywołania destruktorów jest odwrotna w stosunku do konstruktorów - jako pierwszy jest wywoływany destruktor klasy **student**, a po nim destruktor klasy **osoba**

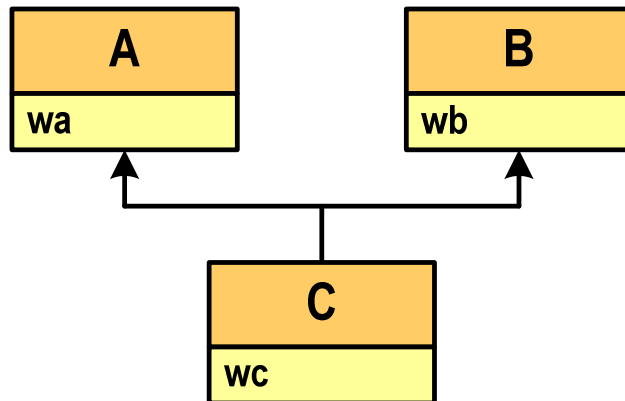
Dziedziczenie wielokrotne

- **dziedziczenie jednokrotne** - klasa tworzona jest na podstawie jednej klasy podstawowej
- **dziedziczenie wielokrotne (wielobazowe)** - klasa tworzona jest na podstawie więcej niż jednej klasy podstawowej (bazowej)



- istnieją języki programowania, w których dziedziczenie wielokrotne nie jest zaimplementowane (np. Java, C#, Object Pascal)

Przykład: dziedziczenie wielokrotne



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

class A
{
    public:
        int wa;
};
```

```
class B
{
    public:
        int wb;
};

class C : public A, public B
{
    public:
        int wc;
};

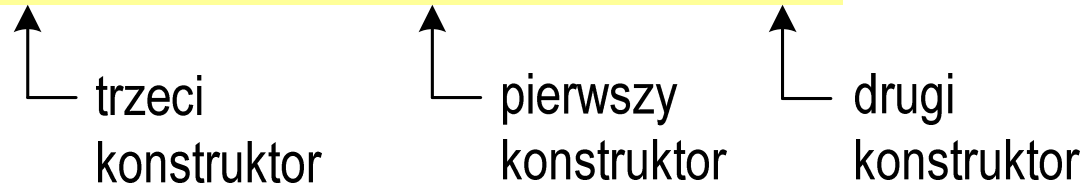
int main(void)
{
    C obiekt;

    obiekt.wa = 1;
    obiekt.wb = 2;
    obiekt.wc = 3;
}
```

Dziedziczenie wielokrotne

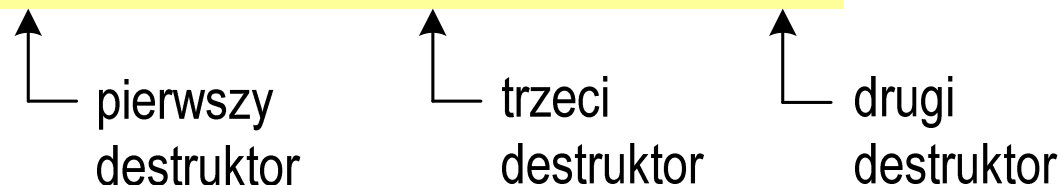
- kolejność wywołania **konstruktorów** dla obiektu klasy pochodnej wynika z kolejności występowania nazw klas bazowych w deklaracji

```
class C : public A, public B
```



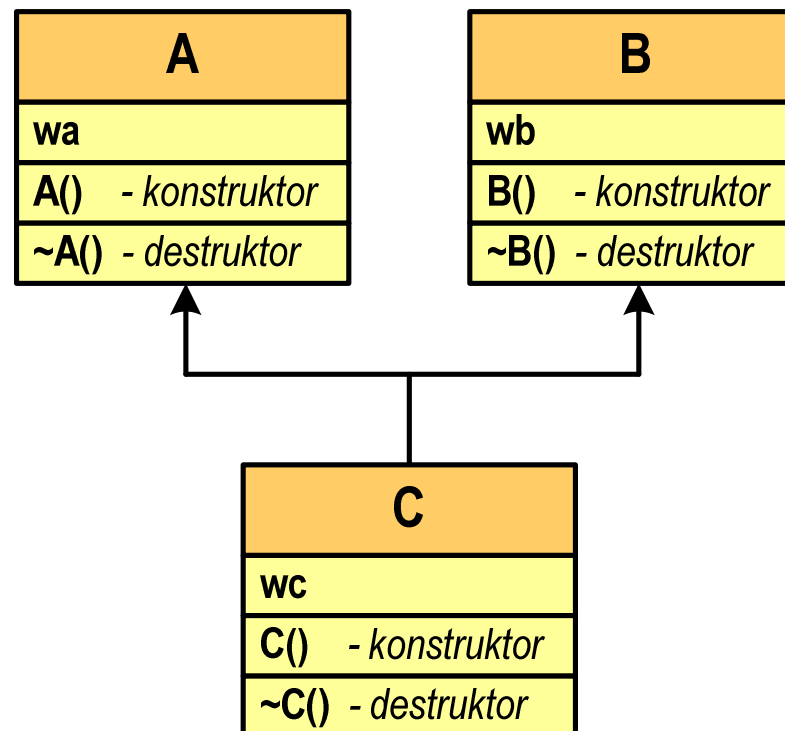
- nie ma znaczenia kolejność umieszczenia konstruktorów klas podstawowych na **liście inicjalizacyjnej** konstruktora klasy pochodnej
- kolejność wywołania **destruktorów** dla obiektu klasy pochodnej jest odwrotna niż konstruktorów

```
class C : public A, public B
```



Przykład: dziedziczenie wielokrotne

- każda klasa (**A**, **B**, **C**) zawiera jedną daną składową, konstruktor, destruktor
- **A**, **B** - klasy podstawowe
- **C** - klasa pochodna



Przykład: dziedziczenie wielokrotne

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

```
class A
```

```
{
```

```
    int wa;
```

```
public:
```

```
    A(int a = 0) : wa(a)
```

```
    {
```

```
        cout << "Konstruktor A()" << endl;
```

```
    }
```

```
    ~A()
```

```
    {
```

```
        cout << "Destruktor ~A()" << endl;
```

```
    }
```

```
};
```

konstruktor klasy A

destruktor klasy A

Przykład: dziedziczenie wielokrotne

```
class B
{
    int wb;
public:
    B(int b = 0) : wb(b)
    {
        cout << "Konstruktor B()" << endl;
    }
    ~B()
    {
        cout << "Destruktor ~B()" << endl;
    }
};
```

konstruktor klasy B

destruktor klasy B

Przykład: dziedziczenie wielokrotne

```
class C : public A, public B
{
    int wc;
public:
    C(int a = 0, int b = 0, int c = 0) : A(a), B(b), wc(c)
    {
        cout << "Konstruktor C()" << endl;
    }
    ~C()
    {
        cout << "Destruktor ~C()" << endl;
    }
};

int main(void)
{
    C obiekt(1, 2, 3);
}
```

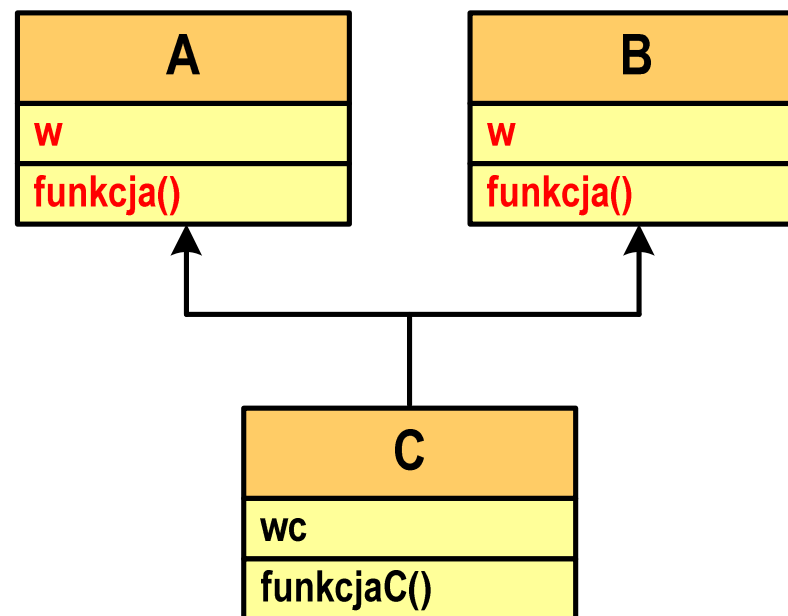
konstruktor klasy C

destruktor klasy c

Konstruktor A()
Konstruktor B()
Konstruktor C()
Destruktor ~C()
Destruktor ~B()
Destruktor ~A()

Problemy w dziedziczeniu wielokrotnym

- podstawowy problem w dziedziczeniu wielokrotnym to możliwość występowania **niejednoznaczności**
- w klasach podstawowych występują dane lub funkcje składowe o takich samych nazwach



Przykład: dziedziczenie wielokrotne (problemy)

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

class A
{
    public:
        int w;
        void funkcja(){};
};

class B
{
    public:
        int w;
        void funkcja(){};
};
```

```
class C : public A, public B
{
    public:
        int wc;
        void funkcjaC(){};
};

int main(void)
{
    C obiekt;

    obiekt.wc = 1;
    obiekt.funkcjaC();
}
```

- w przypadku odwoływania się do danych składowych (**wc**) i funkcji składowych (**funkcjaC**) klasy **C** program skompiluje się i wykona

Przykład: dziedziczenie wielokrotne (problemy)

- próba odwołania się do danej składowej **w** lub wywołania funkcji **funkcja()** spowoduje błąd kompilacji

```
int main(void)
{
    C obiekt;

    obiekt.w = 1;
    obiekt.funkcja();
}
```

```
main.cpp() : error C2385: ambiguous access of 'w'
             could be the 'w' in base 'A'
             or could be the 'w' in base 'B'

main.cpp() : error C2385: ambiguous access of 'funkcja'
             could be the 'funkcja' in base 'A'
             or could be the 'funkcja' in base 'B'

main.cpp() : error C3861: 'funkcja': identifier not found
```

Przykład: dziedziczenie wielokrotne (problemy)

- aby odwołania były jednoznaczne należy zastosować operator zasięgu ::

```
int main(void)
{
    C obiekt;

    obiekt.A::w = 1;
    obiekt.A::funkcja();

    obiekt.B::w = 2;
    obiekt.B::funkcja();
}
```

- w powyższej postaci program skompiluje się i wykona