Dziedziczenie

27.04.2016 & 04.05.2016

W slajdach są materiały zapożyczone z
https://www.ii.uni.wroc.pl/~prz/2012lato/cpp/slajdy/cpp5.ppt

Znajdź błąd

```
class A{
private:
int x, y;
public:
void setValueX();
void setValueY();
};

...

int main(){
A obj;
obj.x = 3;
obj.setValueY(3);
...
}
```

Koncepcja	Realizacja w C++
Abstrakcja	Klasy
Enkapsulacja	Klasy
Ochrona informacji	*public/protected/private*
Polimorfizm	Przeładowanie operatorów, wzorce, metody wirtualne
Dziedziczenie	Dziedziczenie klas

Dziedziczenie

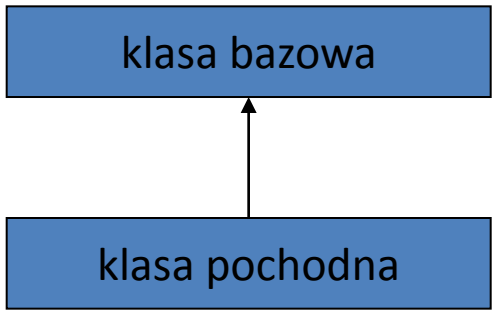
- Dziedziczenie pozwala stworzyć nową klasę przy wykorzystaniu już istniejącej klasy.

- Dziedziczenie to modyfikacja typu polegająca na jego przystosowaniu do określonych warunków – jest to więc rodzaj specjalizacji.

Tekst zapożyczony z https://www.ii.uni.wroc.pl/~prz/2012lato/cpp/slajdy/cpp5.ppt

Dziedziczenie

- Nomenklatura:
  - klasa bazowa (podstawowa albo nadklasa) to klasa, z której dziedziczą inne klasy;
  - klasa pochodna (podklasa) to nowa klasa, która dziedziczy strukturę informacyjną i funkcjonalność z innej klasy.
- Rysunek schematyczny:



Przykład

- Klasa bazowa – Macierz
- Klasa pochodna - MacierzKwadratowa

```
class Macierz {
public:
    Macierz(int n, int m);

    int liczba_wierszy();
    int liczba_kolumn();

    void wypisz();

    void ustaw_wartosc(int wiersz, int kolumna, double wartosc);
    double wartosc(int wiersz, int kolumna);

    void pomnoz_przez(double r);

private:
    int w, k;
    vector< vector<double> > dane;
};
```

```
class MacierzKwadratowa : public Macierz {
public:
    MacierzKwadratowa(int n);

    double wyznacznik();
};
```

Zasada podstawienia Liskov

Funkcje które używają wskaźników lub referencji do klas bazowych, muszą być w stanie używać również obiektów klas dziedziczących po klasach bazowych, bez dokładnej znajomości tych obiektów

Kontrprzykład:

```
void przetwarzajFigurę(Figura& iFigura)
{
  if(typeid(iFigura) == typeid(Prostokąt))
    przetwarzajProstokąt(static_cast<Prostokąt&>(iFigura));
  else if(typeid(iFigura) == typeid(Okrąg))
    przetwarzajOkrąg(static_cast<Okrąg&>(iFigura));

  else if(typeid(iFigura) == typeid(Kwadrat))
    przetwarzajKwadrat(static_cast<Kwadrat&>(iFigura));
}
```

Dostęp do składników

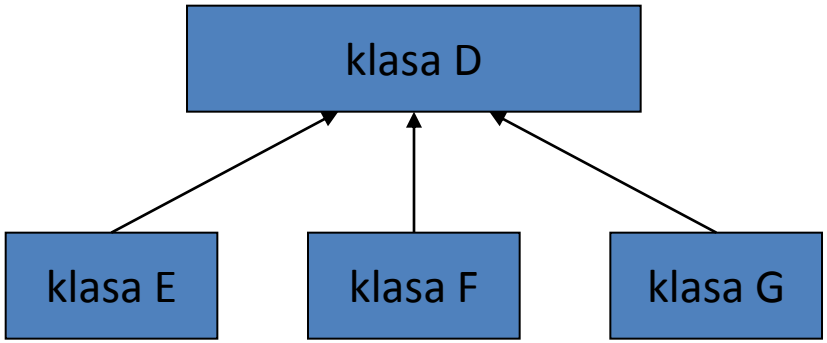
- W klasie pochodnej nie ma dostępu do odziedziczonych składników prywatnych (czyli `private`).
- W klasie pochodnej jest dostęp do odziedziczonych składników nieprywatnych (czyli `protected` i `public`).
- Składniki chronione (czyli `protected`) są dostępne tylko w klasie bieżącej i w klasach pochodnych ale nie na zewnątrz.

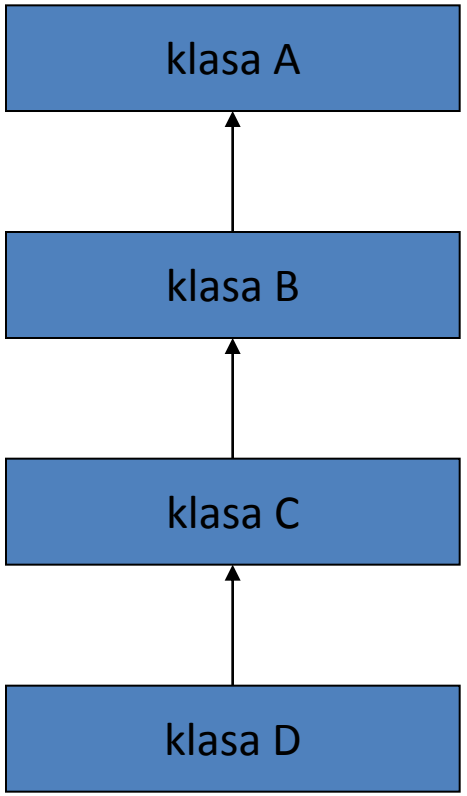
Dostęp do składników

- Klasa pochodna też decyduje o zakresie widoczności odziedziczonych składników nieprywatnych poprzez sposób dziedziczenia (`public`, `protected`, `private`):
  - przy dziedziczeniu publicznym odziedziczone składniki nieprywatne zachowują swój zakres widoczności;
  - przy dziedziczeniu chronionym odziedziczone składniki nieprywatne stają się chronione;
  - przy dziedziczeniu prywatnym odziedziczone składniki nieprywatne stają się prywatne.
- Domyślny sposób dziedziczenia to `private`.

Hierarchia klas

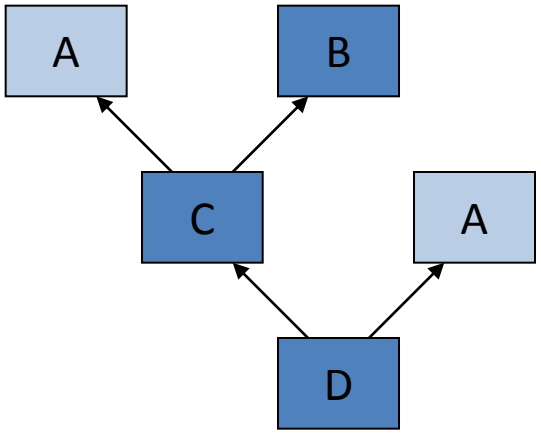
- Dziedziczenie może mieć wiele poziomów.
- Jedna klasa może być klasą bazową dla wielu innych klas.

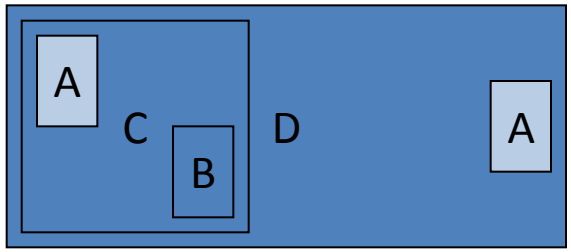




Dziedziczenie wielokrotne (wielodziedziczenie)

- Wielodziedziczenie może prowadzić do wielu skomplikowanych sytuacji: w pojedynczym obiekcie pewna informacja może się wielokrotnie powtórzyć.





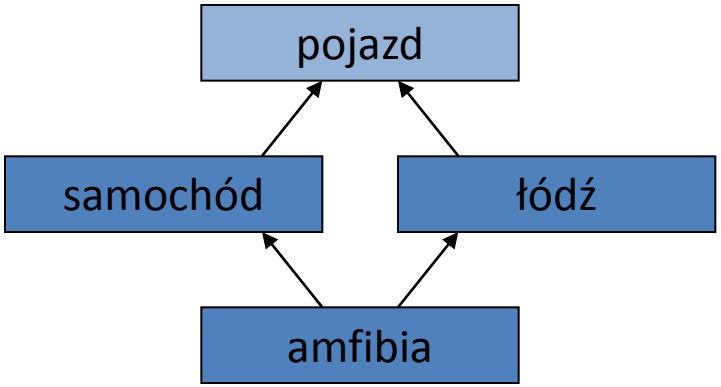
Dziedziczenie wirtualne

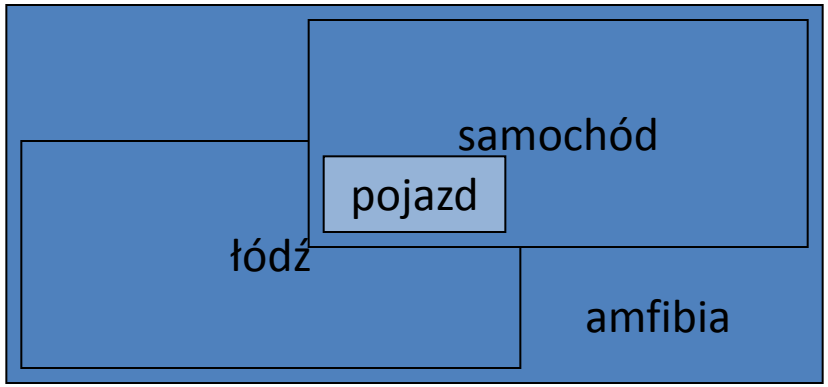
- Dziedziczenie wirtualne może rozwiązać część problemów z dziedziczeniem wielobazowym.
- Dziedziczenie wirtualne powoduje, że pewne informacje występujące wielokrotnie w obiekcie mogą stać się wspólne dla wielu części.
- Dziedziczenie wirtualne deklaruje się słowem `virtual` występującym na liście pochodzenia przed klasą bazową.
- Konstruktor wirtualnej klasy podstawowej jest wywoływany przed konstruktorami jej klas pochodnych.

Dziedziczenie wirtualne

- Przykład dziedziczenia wirtualnego:

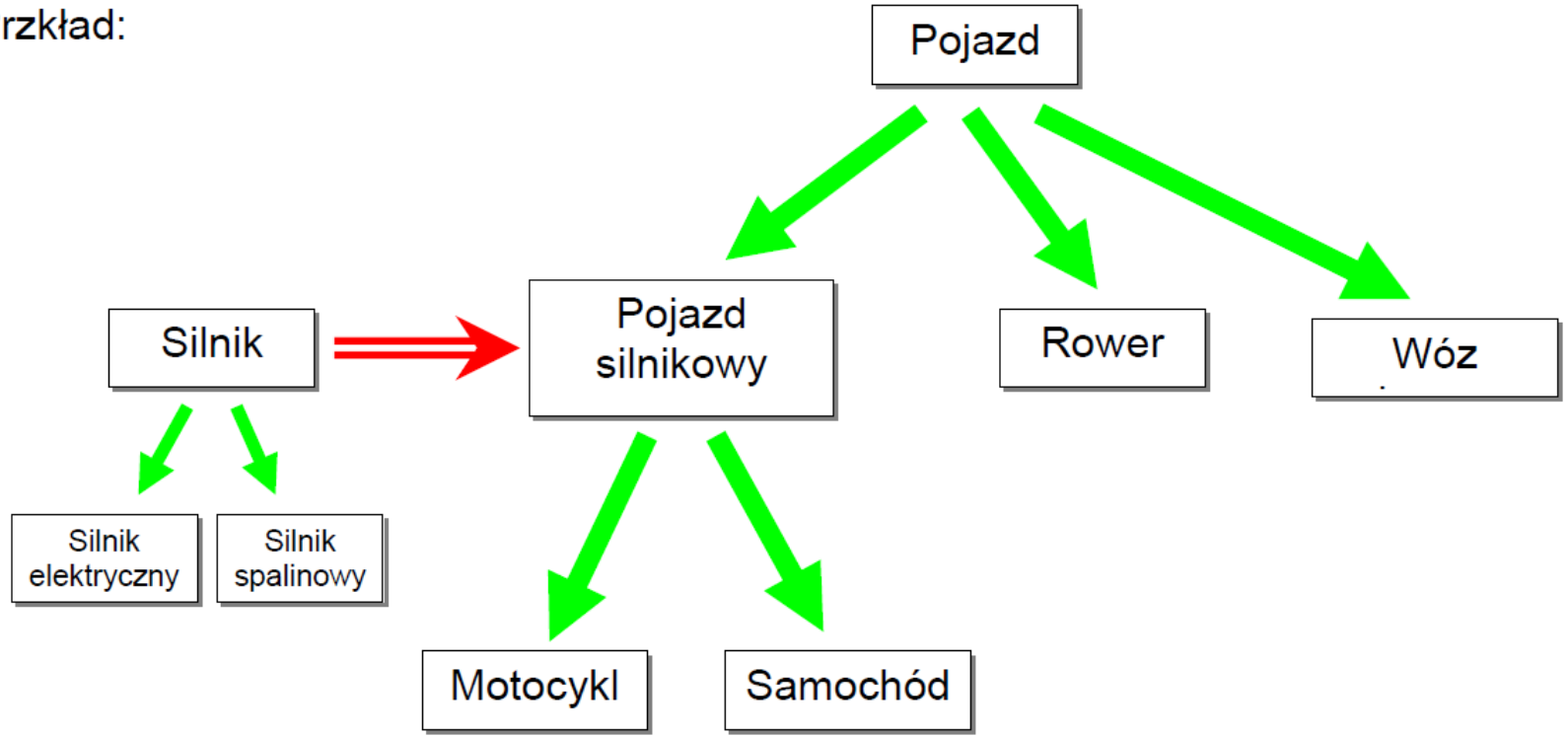
```
class pojazd
    {/*…*/};
class samochód: public virtual pojazd
    {/*…*/};
class łódź: public virtual pojazd
    {/*…*/};
class amfibia: public samochód, public łódź
    {/*…*/};
```





Związki między klasami: „jest” i „zawiera”

Przkład:



Pojazd silnikowy **jest szczególnym rodzajem** **Pojazdu**

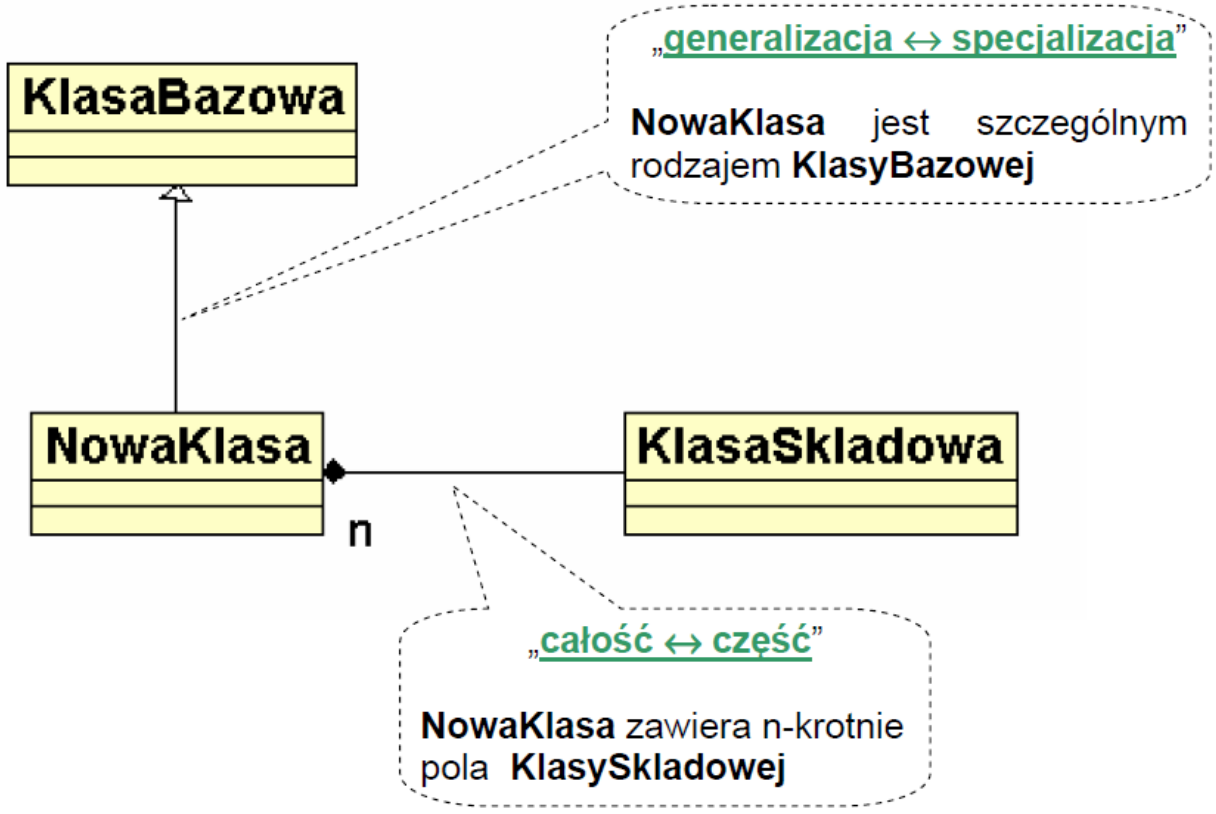
Motocykl **jest szczególnym rodzajem** **Pojazdu silnikowego**

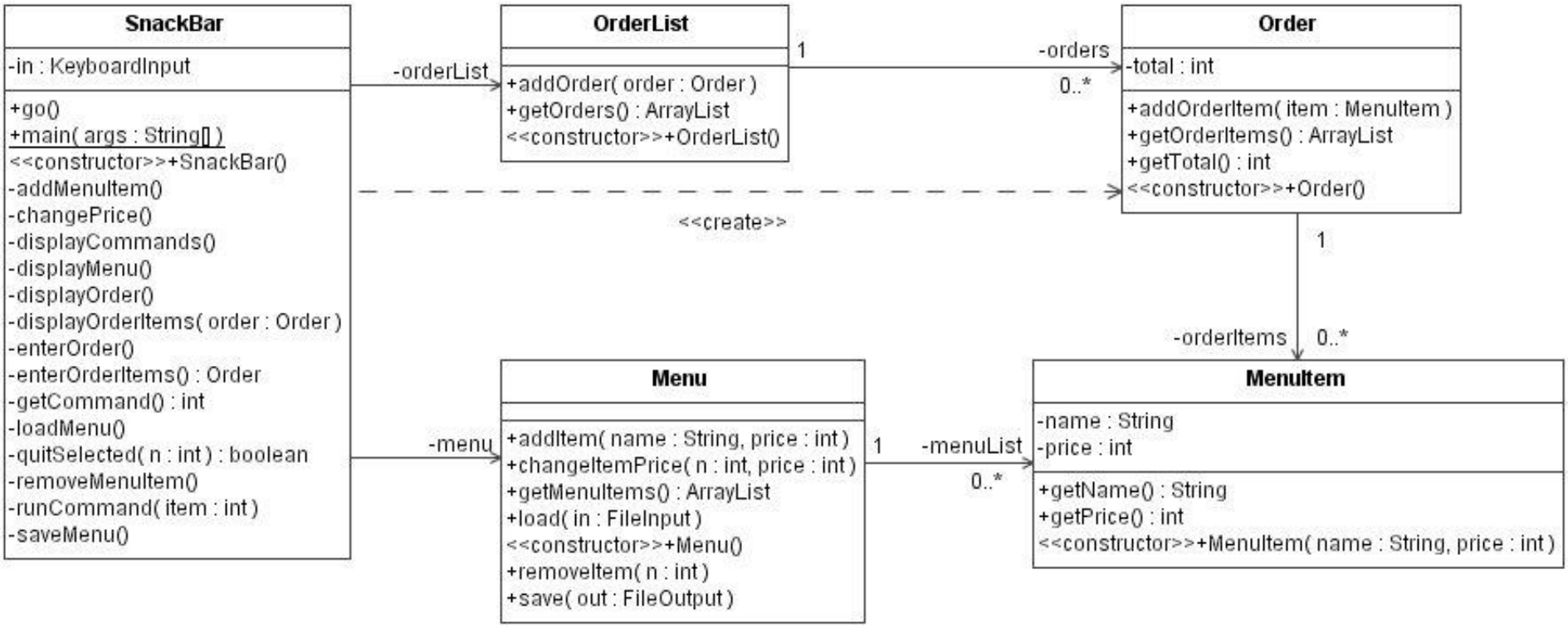
Pojazd silnikowy **zawiera** **Silnik**

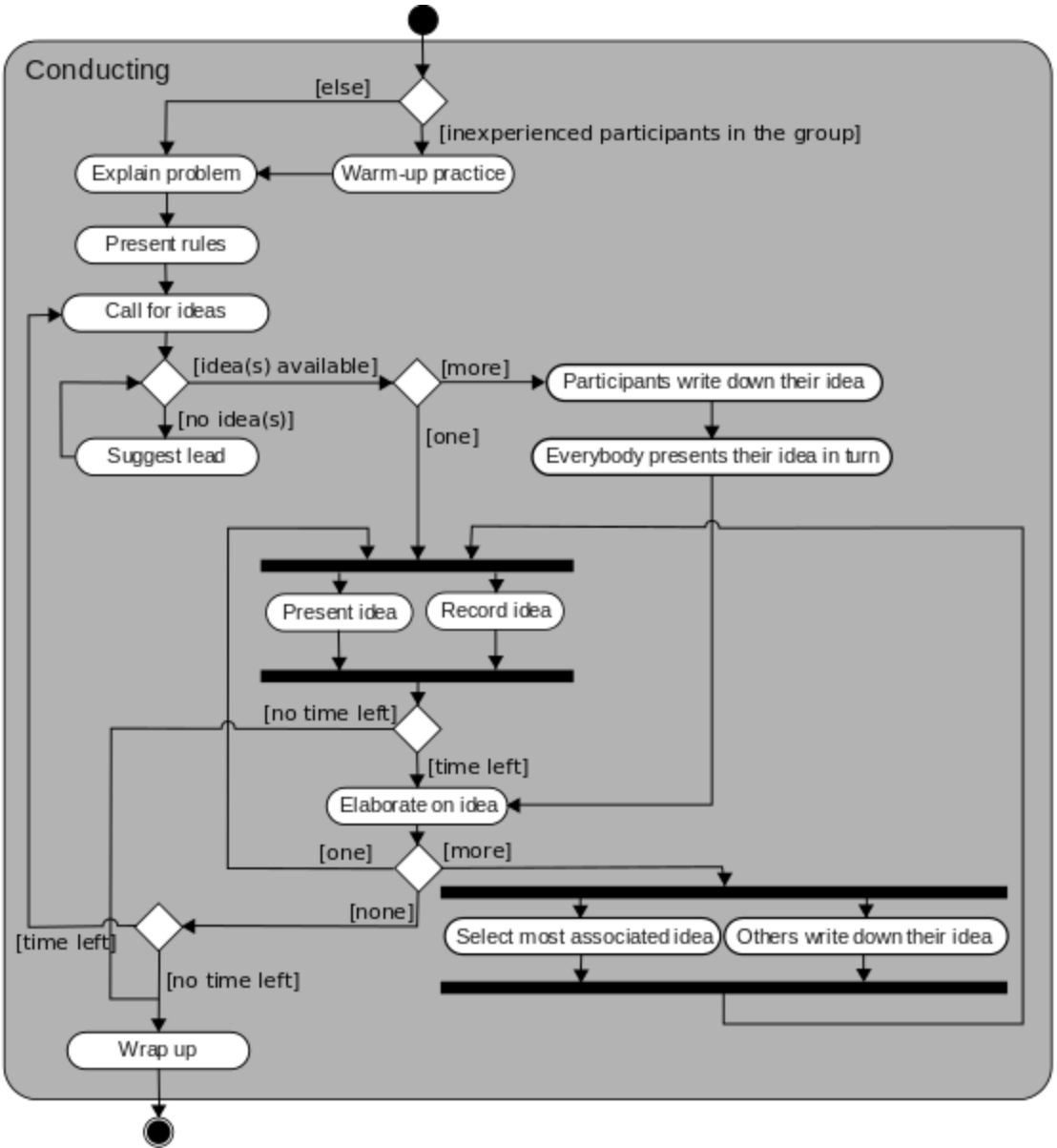
Diagramy klas w języku UML

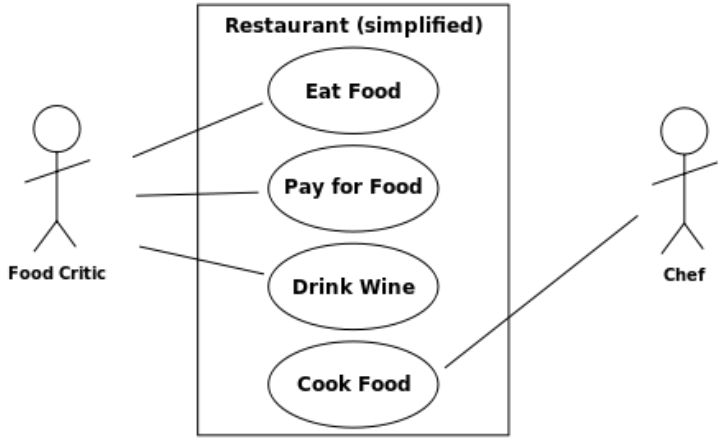
UML (ang. Unified Modeling Language) – zunifikowany język modelowania do tworzenia systemów obiektowo zorientowanych.

Diagram klas pokazuje klasy i zachodzące między nimi relacje.

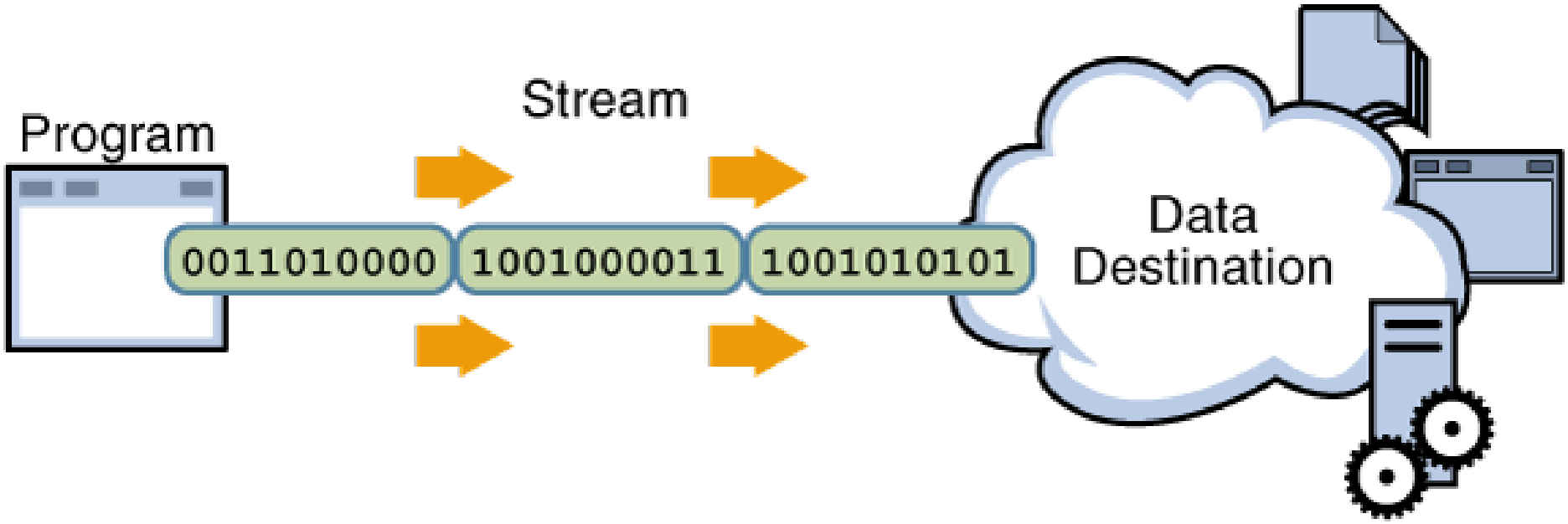



SnackBar
-in : KeyboardInput
+go()
+main(args : String[])
<<constructor>>+SnackBar()
-addMenuItem()
-changePrice()
-displayCommands()
-displayMenu()
-displayOrder()
-displayOrderItems(order : Order)
-enterOrder()
-enterOrderItems() : Order
-getCommand() : int
-loadMenu()
-quitSelected(n : int) : boolean
-removeMenuItem()
-runCommand(item : int)
-saveMenu()
-orderList
OrderList
+addOrder(order : Order)
+getOrders() : ArrayList
<<constructor>>+OrderList()
1
-orders
0..*
Order
-total : int
+addOrderItem(item : MenuItem)
+getOrderItems() : ArrayList
+getTotal() : int
<<constructor>>+Order()
<<create>>
1
-orderItems
0..*
-menu
Menu
+addItem(name : String, price : int)
+changeItemPrice(n : int, price : int)
+getMenuItems() : ArrayList
+load(in : FileInput)
<<constructor>>+Menu()
+removeItem(n : int)
+save(out : FileOutput)
1
-menuList
0..*
MenuItem
-name : String
-price : int
+getName() : String
+getPrice() : int
<<constructor>>+MenuItem(name : String, price : int)


Conducting
[else]
[inexperienced participants in the group]
Explain problem
Warm-up practice
Present rules
Call for ideas
[idea(s) available]
[more]
Participants write down their idea
[no idea(s)]
Suggest lead
[one]
Everybody presents their idea in turn
Present idea
Record idea
[no time left]
[time left]
Elaborate on idea
[one]
[more]
[none]
[time left]
Select most associated idea
Others write down their idea
[no time left]
Wrap up


Restaurant (simplified)
Eat Food
Pay for Food
Drink Wine
Cook Food
Food Critic
Chef

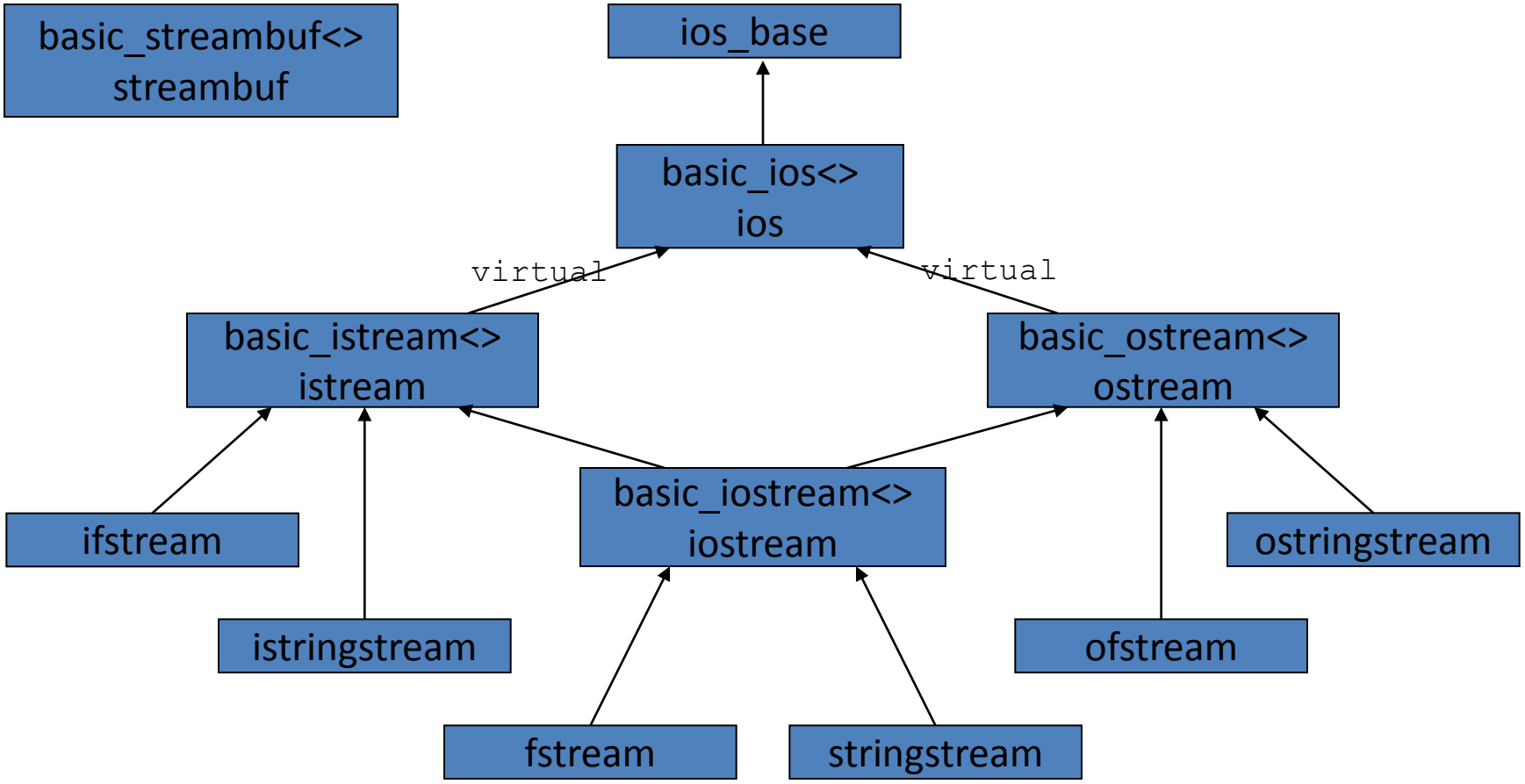
Dwa przykłady – ROOT i Qwt


Program
Stream
0011010000
1001000011
1001010101
Data
Destination

Strumienie

Strumień standardowy
biblioteka:
`<iostream>`

Hierarchia klas strumieniowych realizujących operacje we/wy



Pobieranie danych ze strumienia

operator lub funkcja	Uwagi
>>	Operator >> domyślnie pomija białe znaki. Wczytuje ciąg znaków do pojawiania się kolejnego białego znaku
`get(char* gdzie, int ile, char ogran = '\n')`	Ogranicznik nie jest wyjmowany ze strumienia
`getline(char* gdzie, int ile, char ogran = '\n')`	Ogranicznik jest wyjmowany ze strumienia
`read(char* gdzie, int ile)`	Nie dopisuje na koncu stringu znaku NULL

Zapisywanie danych do strumienia

operator lub funkcja	Uwagi
<<	przekierowanie do strmienia
`put(char znak)`	Wstawia do strumienia jeden znak
`write(char* skąd, int ile)`	Wstawia do strumienia określoną ilość danych

```cpp
#include<iostream>
#include<sstream>
#include<string>


using namespace std;

// wypisanie pierwszej kolumny z 2-kol pliku
int main(int argc, char *argv[])
{
    string line;
    double x,y;
    while(getline(cin, line)){
    istringstream stream(line); // zamiana string na strumien

        stream >> x >> y;
        cout << x << endl;
   }
    return 0;
}
```

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


int main()
{
    string a;
    cout << "Nacisnij Enter aby zakonczyc zapis.\n";
    ofstream f("log.txt");
    cin >> a;
    if (f.good())
    {
        f << a;
        f.close();
    }
    return 0;
}
```

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

int main ()
{
    long x;
    string napis;
    sstringstream ss;
    cout << "Podaj dowolna liczbe calkowita: ";
    cin >> x;
    ss << x;
    napis = ss.str();
    cout << "Dlugosc napisu wynosi "
              << napis.size() << " znakow." << endl;
    return 0;
}
```

```
void wypisz_date(ostream &strumien, int dzien, int miesiac, int rok) {
    strumien << dzien << "." << miesiac << "." << rok << endl;
}
```

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

class Date
{
    int mo, da, yr;
public:
    Date(int m, int d, int y)  {
        mo = m; da = d; yr = y;
    }
    friend ostream& operator<<(ostream& os, const Date& dt);
};
ostream& operator<<(ostream& os, const Date& dt)
{
    os << dt.da << '.' << dt.mo << '.' << dt.yr;
    return os;
}
```

```
int main()
{
    Date dt(5, 6, 92);
    cout << dt;
}
```

```
class Box
{
   public:
       // konstruktor
      Box(double l=2.0, double b=2.0, double h=2.0)
      {
         cout <<"Constructor called." << endl;
         length = l;
         breadth = b;
         height = h;
         // każdy obiekt zwieksza licznik
         objectCount++;
      }
      double Volume()
      {
         return length * breadth * height;
      }
   private:
      double length, breadth, height;
      static int objectCount;

};
// Initialize static member of class Box
int Box::objectCount = 0;
```