Politechnika Wrocławska

Techniki programowania

INP001002Wl

rok akademicki 2018/19

semestr letni

Wykład 4

Karol Tarnowski

karol.tarnowski@pwr.edu.pl

A-1 p. 411B

Politechnika Wrocławska

Plan prezentacji

- Przeciążanie operatorów
- Funkcje zaprzyjaźnione
- Dziedziczenie

Na podstawie:

- A. Allain, *Przewodnik dla początkujących C++*
- S. Prata, *Szkoła programowania C++*

Politechnika Wrocławska

Przeciążanie operatorów

- Przeciążaniu mogą podlegać nie tylko funkcje, ale także operatory.
- Aby przeciążyć operator należy zdefiniować specjalną funkcję – funkcję operatora: **`operator`*`op`*`(`*`lista-parametrów`*`)`**
- Nie można „wymyślać” nowych operatorów.

Przeciążanie operatorów

Przykład

- Załóżmy, że:
  - dysponujemy klasą **`Salesperson`**
  - dla klasy **`Salesperson`** przeciążono operator dodawania, w sposób umożliwiający sumowanie wartości sprzedaży dwóch sprzedawców
  - w programie mamy obiekty **`district2`**, **`sid`**, **`sara`** wszytkie klasy **`Salesperson`**
- Wtedy poprawna będzie instrukcja

```
distric2 = sid + sara;
```

Przeciążanie operatorów

Przykład

- Wtedy poprawna będzie instrukcja
  `distric2 = sid + sara;`
- Kompilator zastąpi standardowy operator odpowiednią funkcją:
  `distric2 = sid.operator+(sara);`

Przeciążanie operatorów

Przykład 2

- Rozważmy klasę **`Time`**

mytime01.h | mytime01.cpp | main.cpp

```
1  #ifndef MYTIME01_H_INCLUDED
2  #define MYTIME01_H_INCLUDED
3
4  class Time{
5      public:
6          Time();
7          Time(int h, int m = 0);
8          void AddMin(int m);
9          void AddHr(int h);
10         void Reset(int h = 0, int m = 0);
11         Time Sum(const Time& t) const;
12         void Show() const;
13     private:
14         int _hours;
15         int _minutes;
16 };
17
18 #endif // MYTIME01_H_INCLUDED
```

Przeciążanie operatorów

Przykład 2

- konstruktory

mytime01.h | **mytime01.cpp** | main.cpp

```
1   #include <iostream>
2   #include "mytime01.h"
3
4   Time::Time()
5       : _hours(0)
6       , _minutes(0)
7   {}
8
9   Time::Time(int h, int m)
10      : _hours(h)
11      , _minutes(m)
12  {}
13
```

Przeciążanie operatorów

Przykład 2

- dodawanie godzin i minut do czasu
- resetowanie czasu

mytime01.h | **mytime01.cpp** | main.cpp

```
14  void Time::AddMin(int m){
15      _minutes += m;
16      _hours += _minutes / 60;
17      _minutes %= 60;
18  }
19
20  void Time::AddHr(int h){
21      _hours += h;
22  }
23
24  void Time::Reset(int h, int m){
25      _hours = h;
26      _minutes = m;
27  }
28
```

Przeciążanie operatorów

Przykład 2

- dodawanie czasów
- wyświetlanie czasu

mytime01.h | **mytime01.cpp** | main.cpp

```
29  Time Time::Sum(const Time& t) const{
30      Time sum;
31      sum._minutes = _minutes + t._minutes;
32      sum._hours   = _hours   + t._hours  + sum._minutes/60;
33      sum._minutes %= 60;
34      return sum;
35  }
36
37  void Time::Show() const{
38      std::cout << _hours << " godzin, " << _minutes << " minut";
39  }
40
```

Przeciążanie operatorów

Przykład 2

mytime01.h | mytime01.cpp | **main.cpp**

```
1  #include <iostream>
2  #include "mytime01.h"
3
4  int main()
5  {
6      using std::cout;
7      using std::endl;
8      Time planning;
9      Time coding(2, 40);
10     Time fixing(5, 55);
11     Time total;
12
13     cout << "czas planowania = ";
14     planning.Show();
15     cout << endl;
16
17     cout << "czas kodowania = ";
18     coding.Show();
19     cout << endl;
20
21     cout << "czas poprawiania = ";
22     fixing.Show();
23     cout << endl;
```

Przeciążanie operatorów

Przykład 2

mytime01.h | mytime01.cpp | **main.cpp**

```
25      total = coding.Sum(fixing);
26      cout << "razem (coding.sum(fixing)) = ";
27      total.Show();
28      cout << endl;
29
30      return 0;
31  }
```

```
czas planowania = 0 godzin, 0 minut
czas kodowania = 2 godzin, 40 minut
czas poprawiania = 5 godzin, 55 minut
razem (coding.sum(fixing)) = 8 godzin, 35 minut
```

Przeciążanie operatorów

Przykład 3

- Metoda **`Sum()`** zastąpiona funkcją operatorową

mytime02.h | mytime02.cpp | main.cpp

```
1  #ifndef MYTIME02_H_INCLUDED
2  #define MYTIME02_H_INCLUDED
3
4  class Time{
5      public:
6          Time();
7          Time(int h, int m = 0);
8          void AddMin(int m);
9          void AddHr(int h);
10         void Reset(int h = 0, int m = 0);
11         Time operator+(const Time& t) const;
12         void Show() const;
13     private:
14         int _hours;
15         int _minutes;
16 };
17
18 #endif // MYTIME02_H_INCLUDED
19
```

Przeciążanie operatorów

Przykład 3

- Metoda **`Sum()`** zastąpiona funkcją operatorową

mytime02.h | main.cpp | **mytime02.cpp**

```
29  Time Time::operator+(const Time& t) const{
30      Time sum;
31      sum._minutes = _minutes + t._minutes;
32      sum._hours   = _hours   + t._hours  + sum._minutes/60;
33      sum._minutes %= 60;
34      return sum;
35  }
```

Przeciążanie operatorów

Przykład 3

- Wywołanie

mytime02.h | mytime02.cpp | **main.cpp**

```
25      total = coding + fixing;
26      cout << "razem (coding + fixing) = ";
27      total.Show();
28      cout << endl;
29
30      Time morefixing(3, 28);
31      cout << "kolejne poprawki = ";
32      morefixing.Show();
33      cout << endl;
34      total = morefixing.operator+(total);
35      cout << "razem (morefixing.operator+(total)) = ";
36      total.Show();
37      cout << endl;
```

```
czas planowania = 0 godzin, 0 minut
czas kodowania = 2 godzin, 40 minut
czas poprawiania = 5 godzin, 55 minut
razem (coding + fixing) = 8 godzin, 35 minut
kolejne poprawki = 3 godzin, 28 minut
razem (morefixing.operator+(total)) = 12 godzin, 3 minut
```

Politechnika Wrocławska

Przeciążanie operatorów

Ograniczenie

- Przeciążony operator musi przyjmować przynajmniej jeden operand typu własnego
- Nie można korzystać z operatora w sposób naruszający jego składnię (np. zmieniać liczbę operandów)
- Nie można definiować własnych operatorów
- Nie wszystkie operatory można przeciążać

Przeciążanie operatorów

Przykład 4

mytime03.h | mytime03.cpp | main.cpp

```
1   #ifndef MYTIME03_H_INCLUDED
2   #define MYTIME03_H_INCLUDED
3
4   class Time{
5       public:
6           Time();
7           Time(int h, int m = 0);
8           void AddMin(int m);
9           void AddHr(int h);
10          void Reset(int h = 0, int m = 0);
11          Time operator+(const Time& t) const;
12          Time operator-(const Time& t) const;
13          Time operator*(double n) const;
14          void Show() const;
15      private:
16          int _hours;
17          int _minutes;
18  };
19
20  #endif // MYTIME03_H_INCLUDED
21
```

Przeciążanie operatorów

Przykład 4

mytime03.h | **mytime03.cpp** | main.cpp

```
37  Time Time::operator-(const Time& t) const{
38      Time diff;
39      int tot1, tot2;
40      tot1 = t._minutes + 60*t._hours;
41      tot2 =   _minutes + 60*  _hours;
42      diff._minutes = (tot2 - tot1)%60;
43      diff._hours   = (tot2 - tot1)/60;
44      return diff;
45  }
46
47  Time Time::operator*(double mult) const{
48      Time result;
49      long totalMinutes = _hours*mult*60 + _minutes*mult;
50      result._minutes = totalMinutes%60;
51      result._hours   = totalMinutes/60;
52      return result;
53  }
```

Przeciążanie operatorów

Przykład 4

mytime03.h | mytime03.cpp | **main.cpp**

```
22      total = weeding + waxing;
23      cout << "razem czas pracy = ";
24      total.Show();
25      cout << endl;
26
27      diff = weeding - waxing;
28      cout << "czas pielenia - czas woskowania = ";
29      diff.Show();
30      cout << endl;
31
32      adjusted = total * 1.5;
33      cout << "czas pracy z poprawka na przerwy = ";
34      adjusted.Show();
35      cout << endl;
36
37      return 0;
38  }
```

Politechnika Wrocławska

Przeciążanie operatorów

- Wywołanie

```
adjusted = total * 1.5;
```

jest prawidłowe, ale

```
adjusted = 1.5 * total;
```

nie zadziała

Politechnika Wrocławska

Funkcje zaprzyjaźnione

- Funkcja zaprzyjaźniona nie jest metodą klasy, ale ma uprawnienia zrównujące ją z metodami klasy.

Funkcje zaprzyjaźnione

- w deklaracji klasy

```
friend Time operator*(double m, const Time& t);
```

- definicja funkcji

```
Time operator*(double m, const Time& t){
    Time result;
    long totalMinutes = t._hours*m*60 + t._minutes*m;
    result._minutes = totalMinutes%60;
    result._hours   = totalMinutes/60;
    return result;
}
```

Politechnika Wrocławska

Funkcje zaprzyjaźnione

- możliwa jest inna definicja funkcji

```
Time operator*(double m, const Time& t){
    return t * m;
}
```

- tak zdefiniowana funkcja nie wymagałaby zaprzyjaźniania

Politechnika Wrocławska

Funkcje zaprzyjaźnione

- przeciążony operator<< do wypisywania

```
void operator<<(ostream& out, const Time& t){
     out << t._hours << " godzin, ";
     out << t._minutes << " minut";
}
```

- wykorzystując ten operator możemy zapisać

```
cout << total;
```

Politechnika Wrocławska

Funkcje zaprzyjaźnione

- przeciążony operator<< do wypisywania

```
ostream& operator<<(ostream& out, const Time& t){
    out << t._hours << " godzin, ";
    out << t._minutes << " minut";
    return out;
}
```

- wykorzystując ten operator możemy zapisać

```
cout << "czas razem" << total << endl;
```

Politechnika Wrocławska

Funkcje zaprzyjaźnione

- zasadniczo przeciążony operator<< ma postać

```
ostream& operator<<(ostream& out, const klasa& obj){
    return out << ... ; //wyświetlenie obiektu
}
```

Politechnika Wrocławska

Dziedziczenie

- Dziedziczenie pozwala dodawać nowe możliwości (nowe metody) do istniejącej klasy
- Dziedziczenie pozwala dodawać nowe dane do klasy
- Dziedziczenie pozwala zmieniać działanie metod klasy

Politechnika Wrocławska

Dziedziczenie

- Prosta klasa bazowa:
  - przechowuje imię i nazwisko oraz informację o tym, czy gracz ma stół pingpongowy

Politechnika Wrocławska

Dziedziczenie

TableTennisPlayer.h

```
1  #ifndef TABLETENNISPLAYER_H
2  #define TABLETENNISPLAYER_H
3  #include <string>
4
5  class TableTennisPlayer
6  {
7      public:
8          TableTennisPlayer(const std::string & fn = "brak", \
9                            const std::string & ln = "brak", \
10                           bool ht = false);
11         void name() const;
12         bool hasTable() const { return _has_table; };
13         void resetTable(bool v) { _has_table = v; };
14
15     private:
16         std::string _first_name;
17         std::string _last_name;
18         bool   _has_table;
19 };
20
21 #endif // TABLETENNISPLAYER_H
```

Politechnika Wrocławska

Dziedziczenie

TableTennisPlayer.h | **TableTennisPlayer.cpp**

```
1   #include "TableTennisPlayer.h"
2   #include <iostream>
3   using std::string;
4   using std::cout;
5
6   TableTennisPlayer::TableTennisPlayer(const string & fn, const string & ln, bool ht)
7       : _first_name(fn) \
8       , _last_name(ln)  \
9       , _has_table(ht) {}
10
11  void TableTennisPlayer::name() const{
12      cout << _last_name << ", " << _first_name;
13  }
14
```

Politechnika Wrocławska

Dziedziczenie

TableTennisPlayer.h | TableTennisPlayer.cpp | **main.cpp**

```
1  #include <iostream>
2  #include "include\TableTennisPlayer.h"
3
4  int main()
5  {
6      TableTennisPlayer player1("Jacek","Pogodny", true);
7      TableTennisPlayer player2("Teresa","Bogatko", false);
8
9      player1.name();
10     if(player1.hasTable())
11         std::cout << ": posiada stol." << std::endl;
12     else
13         std::cout << ": nie posiada stolu." << std::endl;
14
15     player2.name();
16     if(player2.hasTable())
17         std::cout << ": posiada stol." << std::endl;
18     else
19         std::cout << ": nie posiada stolu." << std::endl;
20
21     return 0;
22 }
23
```

Politechnika Wrocławska

Dziedziczenie

- Klasa pochodna:
  - dodatkowa informacja o punktach rankingowych

```
class RatedPlayer : public TableTennisPlayer
{
...
};
```

- Obiekt klasy pochodnej zawiera w sobie dane składowe typu bazowego.
- Klasa pochodna dziedziczy implementację klasy bazowej.

Politechnika Wrocławska

Dziedziczenie

- Klasa pochodna potrzebuje własnego konstruktora.
- W klasie pochodnej można dodać nowe pola i metody.

Politechnika Wrocławska

Dziedziczenie

TableTennisPlayer.h | TableTennisPlayer.cpp | main.cpp | **RatedPlayer.h** | RatedPlayer.cpp

```
1  #ifndef RATEDPLAYER_H
2  #define RATEDPLAYER_H
3
4  #include <TableTennisPlayer.h>
5
6
7  class RatedPlayer : public TableTennisPlayer
8  {
9      public:
10         RatedPlayer(unsigned int r = 0, \
11                     const string & fn = "brak", \
12                     const string & ln = "brak", \
13                     bool ht = false);
14         RatedPlayer(unsigned int r, \
15                     const TableTennisPlayer & tp);
16         unsigned int rating() const { return _rating; };
17         void resetRating(unsigned int r){ _rating = r; };
18
19     private:
20         unsigned int _rating;
21 };
22
23 #endif // RATEDPLAYER_H
```

Politechnika Wrocławska

Dziedziczenie

- Klasa pochodna nie ma dostępu do prywatnych składowych klasy bazowej, do ich inicjalizacji należy wykorzystać listę inicjalizacyjną

Politechnika Wrocławska

Dziedziczenie

TableTennisPlayer.h | TableTennisPlayer.cpp | main.cpp | RatedPlayer.h | **RatedPlayer.cpp**

```
1   #include "RatedPlayer.h"
2
3   RatedPlayer::RatedPlayer(unsigned int r, \
4     const string & fn, const string & ln, bool ht)
5     : TableTennisPlayer(fn, ln, ht)
6   {
7       _rating = r;
8   }
9
10  /*
11  pominięcie konstruktora klasy bazowej powodowałoby
12  wywołanie konstruktura domyślnego
13
14  RatedPlayer::RatedPlayer(unsigned int r, \
15    const string & fn, const string & ln, bool ht)
16  {
17      _rating = r;
18  }
19  */
```

Politechnika Wrocławska

Dziedziczenie

TableTennisPlayer.h | TableTennisPlayer.cpp | main.cpp | RatedPlayer.h | *RatedPlayer.cpp

```
21  RatedPlayer::RatedPlayer(unsigned int r, const TableTennisPlayer & tp)
22      : TableTennisPlayer(tp)
23  {
24      _rating = r;
25  }
26
27  /*
28  powyższy konstruktor jest równoważny następującemu
29  RatedPlayer::RatedPlayer(unsigned int r, const TableTennisPlayer & tp)
30      : TableTennisPlayer(tp)
31      , _rating(r)
32  {
33  }*/
```

Politechnika Wrocławska

Dziedziczenie

TableTennisPlayer.h | TableTennisPlayer.cpp | **main.cpp** | RatedPlayer.h | *RatedPlayer.cpp

```
1   #include <iostream>
2   #include "include\TableTennisPlayer.h"
3   #include "include\RatedPlayer.h"
4
5   int main()
6   {
7       TableTennisPlayer  player1("Teresa", "Bogatko", false);
8       RatedPlayer       rplayer1(1140, "Maciej", "Kaczkowski", true);
9
10      rplayer1.name();
11      if(rplayer1.hasTable())
12          std::cout << ": posiada stol." << std::endl;
13      else
14          std::cout << ": nie posiada stolu." << std::endl;
15
16      player1.name();
17      if(player1.hasTable())
18          std::cout << ": posiada stol." << std::endl;
19      else
20          std::cout << ": nie posiada stolu." << std::endl;
21
22      std::cout << "Nazwisko i imie: ";
23      rplayer1.name();
24      std::cout << "; Ranking: " << rplayer1.rating() << std::endl;
25
26      RatedPlayer rplayer2(1212, player1);
27      std::cout << "Nazwisko i imie: ";
28      rplayer2.name();
29      std::cout << "; Ranking: " << rplayer2.rating() << std::endl;
30
31      return 0;
32  }
33
```

Politechnika Wrocławska

Dziedziczenie

- Wskaźniki klasy bazowej mogą pokazywać obiekty klasy pochodnej.
- Referencja klasy bazowej może odnosić się do obiektu klasy pochodnej.

Politechnika Wrocławska

Dziedziczenie

main.cpp

```
 6
 7  int main()
 8  {
 9      RatedPlayer       rplayer(1140, "Maciej", "Kaczkowski", true);
10      TableTennisPlayer & rt = rplayer;
11      TableTennisPlayer * pt = &rplayer;
12      rt.name();
13      std::cout << std::endl;
14      pt->name();
15      std::cout << std::endl;
16
17      TableTennisPlayer  player("Teresa", "Bogatko", false);
18      //RatedPlayer & rr = player;  NIEDOZWOLONE
19      //RatedPlayer * pr = &player; NIEDOZWOLONE
20
```

Politechnika Wrocławska

Dziedziczenie

- W konsekwencji funkcje pobierające jako argument referencję do klasy bazowej działają także dla obiektów klasy pochodnej.

Politechnika Wrocławska

Dziedziczenie

main.cpp

```
36  void Show(const TableTennisPlayer & rt){
37      std::cout << "Nazwisko i imie: ";
38      rt.name();
39      std::cout << "\nStol: ";
40      if( rt.hasTable() )
41          std::cout << "tak\n";
42      else
43          std::cout << "nie\n";
44  }
```

```
RatedPlayer rplayer(1140, "Maciej", "Kaczkowski", true);
TableTennisPlayer  player("Teresa", "Bogatko", false);
Show(player);
Show(rplayer);
```

Politechnika Wrocławska

Dziedziczenie

- Podobnie można inicjalizować obiekty klasy bazowej za pomocą obiektu klasy pochodnej

main.cpp

```
24      RatedPlayer olaf1(1840, "Olaf", "Bochenek", true);
25      TableTennisPlayer olaf2(olaf1);
26      //TableTennisPlayer(const RatedPlayer& );        nie ma takiego konstruktora
27      //TableTennisPlayer(const TableTennisPlayer& );  ale jest taki niejawny konstruktor kopiujący
28
29      TableTennisPlayer winner;
30      winner = olaf1;  //przypisuje obiekt klasy pochodnej do obiektu klasy bazowej
31      //TableTennisPlayer & operator=(const TableTennisPlayer &) const;
32
```

Politechnika Wrocławska

Dziedziczenie

- Rachunek bankowy
  - imię i nazwisko klienta,
  - numer rachunku,
  - saldo.
- Operacje na rachunku:
  - tworzenie rachunku,
  - wpłata,
  - wypłata,
  - wyświetlenie.

Politechnika Wrocławska

Dziedziczenie

- Rachunek bankowy rozszerzony o:
  - limit debetu,
  - stopa procentowa,
  - zadłużenie.
- Operacje na rachunku:
  - wypłata (obsługuje powstanie debetu),
  - wyświetlenie (w tym dodatkowych informacji).

Politechnika Wrocławska

Dziedziczenie

main.cpp | Account.h

```
1  #ifndef ACCOUNT_H
2  #define ACCOUNT_H
3  #include<string>
4
5  class Account
6  {
7      public:
8          Account(const std::string & s = "brak", long an = -1, double bal = 0.0);
9          void deposit(double amount);
10         virtual void withdraw(double amount);
11         double balance() const;
12         virtual void viewAccount() const;
13         virtual ~Account() {};
14
15     private:
16         std::string _full_name;
17         long _account_number;
18         double _balance;
19 };
20
```

Politechnika Wrocławska

Dziedziczenie

main.cpp | Account.h

```
21  class AccountPlus : public Account
22  {
23      public:
24          AccountPlus(const std::string & s = "brak", long an = -1, double bal = 0.0,
25                      double ml = 2000, double r = 0.11125);
26          AccountPlus(const Account & a, double ml = 2000, double r = 0.11125);
27          virtual void withdraw(double amount);
28          virtual void viewAccount() const;
29          void resetMax(double m){_max_loan = m; };
30          void resetRate(double r){_rate = r; };
31          void resetOwes(){_owes = 0;};
32
33      private:
34          double _max_loan;
35          double _rate;
36          double _owes;
37  };
38
39
40  #endif // ACCOUNT_H
```

Politechnika Wrocławska

Dziedziczenie

- W obu klasach zadeklarowano metody **`withdraw()`**, **`viewAccount()`**
- Nowe słowo kluczowe **`virtual`**
- W klasie **`Account`** jest zadeklarowany także destruktor wirtualny

Politechnika Wrocławska

Dziedziczenie

main.cpp | Account.h | **Account.cpp**

```
13  Account::Account(const string& s, long an, double bal)
14      : _full_name(s)
15      , _account_number(an)
16      , _balance(bal)
17  {}
18
19  void Account::deposit(double amount){
20      if( amount < 0 )
21          cout << "Nie mozna wplacic ujemnej kwoty; "
22               << "wplata anulowana." << endl;
23      else
24          _balance += amount;
25  }
```

Politechnika Wrocławska

Dziedziczenie

main.cpp | Account.h | **Account.cpp**

```
27  void Account::withdraw(double amount){
28      format initial_state = setFormat();
29      precis prec = cout.precision(2);
30      if( amount < 0 )
31          cout << "Nie mozna wyplacic ujemnej kwoty; "
32               << "wyplata anulowana." << endl;
33      else if( amount < _balance )
34          _balance -= amount;
35      else
36          cout << "Zadana suma " << amount
37               << " PLN przekracza dostepne srodki." << endl
38               << "Wyplata anulowana." << endl;
39      restore(initial_state, prec);
40  }
41
42  double Account::balance() const{
43      return _balance;
44  }
45
46  void Account::viewAccount() const{
47      format initial_state = setFormat();
48      precis prec = cout.precision(2);
49      cout << "Klient: " << _full_name << endl;
50      cout << "Numer rachunku: " << _account_number << endl;
51      cout << "Stan konta: " << _balance << " PLN" << endl;
52      restore(initial_state, prec);
53  }
```

Politechnika Wrocławska

Dziedziczenie

main.cpp | Account.h | **Account.cpp**

```
55  AccountPlus::AccountPlus(const string& s, long an, double bal,
56                           double ml, double r)
57      : Account(s, an, bal)
58      , _max_loan(ml)
59      , _rate(r)
60      , _owes(0)
61  {}
62
63  AccountPlus::AccountPlus(const Account & a, double ml, double r)
64      : Account(a)
65      , _max_loan(ml)
66      , _rate(r)
67      , _owes(0)
68  {}
```

Politechnika Wrocławska

Dziedziczenie

main.cpp | Account.h | **Account.cpp**

```
70  void AccountPlus::withdraw(double amount){
71      format initial_state = setFormat();
72      precis prec = cout.precision(2);
73
74      double bal = balance();
75      if( amount < bal )
76          Account::withdraw(amount);
77      else if( amount < bal + _max_loan - _owes ){
78          double advance = amount - bal;
79          _owes += advance * (1.0 + _rate);
80          cout << "Zadluzenie faktyczne: " << advance << " PLN" << endl;
81          cout << "Odsetki: " << advance * _rate << " PLN" << endl;
82          deposit(advance);
83          Account::withdraw(amount);
84      }
85      else
86          cout << "Przekroczony limit debetu. Operacja anulowana" << endl;
87      restore(initial_state, prec);
88  }
89
90  void AccountPlus::viewAccount() const{
91      format initial_state = setFormat();
92      precis prec = cout.precision(2);
93      Account::viewAccount();
94      cout << "Limit debetu: " << _max_loan << endl;
95      cout << "Kwota zadluzenia: " << _owes << endl;
96      cout.precision(3);
97      cout << "Stopa oprocentowania: " << 100*_rate << "%" << endl;
98      restore(initial_state, prec);
99  }
```

Politechnika Wrocławska

Dziedziczenie

- Konstruktory klasy pochodnej używają list inicjalizacyjnych
- Metoda **`AccountPlus::viewAccount()`** wywołuje metodę **`Account::viewAccount()`**
- Metoda **`AccountPlus::withdraw`** wywołuje, w zależności od sytuacji, różne metody

Politechnika Wrocławska

Dziedziczenie

main.cpp | Account.h | Account.cpp

```
1  #include <iostream>
2  #include "Account.h"
3
4  int main()
5  {
6      using std::cout;
7      using std::endl;
8
9      Account piggy("Bonifacy Kot", 381299, 12000.00);
10     AccountPlus hoggy("Horacy Biedronka", 382288, 9000.00);
11     piggy.viewAccount();
12     cout << endl;
13     hoggy.viewAccount();
14     cout << endl;
15
16     cout << "Wplata 3000 PLN na rachunek pana Biedronki." << endl;
17     hoggy.deposit(3000.00);
18     cout << "Nowy stan konta: " << hoggy.balance() << " PLN" << endl;
19     cout << "Wyplata 12600 PLN z rachunku pana Kota." << endl;
20     piggy.withdraw(12600.00);
21     cout << "Stan konta Kota: " << piggy.balance() << " PLN" << endl;
22     cout << "Wyplata 12600 PLN z rachunku pana Biedronki." << endl;
23     hoggy.withdraw(12600.00);
24     hoggy.viewAccount();
25
26
27     return 0;
28 }
29
```

Politechnika Wrocławska

Dziedziczenie

```
Account * p_clients[CLIENTS];
...
for(int i=0; i < CLIENTS; i++){
     p_clients[i]->viewAccount();
     cout << endl;
}
for(int i=0; i < CLIENTS; i++){
     delete p_clients[i];
}
```

Politechnika Wrocławska

Dziedziczenie

```
class Account{
    protected:
        double balance;
...
};
```

Politechnika Wrocławska

Dziedziczenie

- Abstrakcyjna klasa bazowa **`AccountABC`**

Politechnika Wrocławska

Dziedziczenie

main.cpp | AccountABC.h | AccountABC.cpp

```
#ifndef ACCOUNTABC_H
#define ACCOUNTABC_H
#include <iostream>
#include <string>

class AccountABC
{
    public:
        AccountABC(const std::string & s = "brak", long an = -1, double bal = 0.0);
        void deposit(double amount);
        virtual void withdraw(double amount) = 0;
        double balance() const {return _balance; };
        virtual void viewAccount() const = 0;
        virtual ~AccountABC() {};

    protected:
        struct Formatting{
            std::ios_base::fmtflags flag;
            std::streamsize pr;
        };
        const std::string & fullName() const {return _full_name; }
        long accountNumber() const { return _account_number; }
        Formatting setFormat() const;
        void restore(Formatting & f) const;

    private:
        std::string _full_name;
        long _account_number;
        double _balance;
};
```

Politechnika Wrocławska

Dziedziczenie

main.cpp | AccountABC.h | AccountABC.cpp

```
32  class Account : public AccountABC
33  {
34      public:
35          Account(const std::string & s = "brak", long an = -1, double bal = 0.0)
36              : AccountABC(s, an, bal){}
37          virtual void withdraw(double amount);
38          virtual void viewAccount() const;
39          virtual ~Account() {};
40  };
41
42  class AccountPlus : public AccountABC
43  {
44      public:
45          AccountPlus(const std::string & s = "brak", long an = -1, double bal = 0.0,
46                      double ml = 2000, double r = 0.11125);
47          AccountPlus(const Account & a, double ml = 2000, double r = 0.11125);
48          virtual void withdraw(double amount);
49          virtual void viewAccount() const;
50          void resetMax(double m){_max_loan = m; };
51          void resetRate(double r){_rate = r; };
52          void resetOwes(){_owes = 0;};
53
54      private:
55          double _max_loan;
56          double _rate;
57          double _owes;
58  };
59
60
61  #endif // ACCOUNTABC_H
```

Politechnika Wrocławska

Dziedziczenie

main.cpp | AccountABC.h | AccountABC.cpp

```
32  class Account : public AccountABC
33  {
34      public:
35          Account(const std::string & s = "brak", long an = -1, double bal = 0.0)
36              : AccountABC(s, an, bal){}
37          virtual void withdraw(double amount);
38          virtual void viewAccount() const;
39          virtual ~Account() {};
40  };
41
42  class AccountPlus : public AccountABC
43  {
44      public:
45          AccountPlus(const std::string & s = "brak", long an = -1, double bal = 0.0,
46                      double ml = 2000, double r = 0.11125);
47          AccountPlus(const Account & a, double ml = 2000, double r = 0.11125);
48          virtual void withdraw(double amount);
49          virtual void viewAccount() const;
50          void resetMax(double m){_max_loan = m; };
51          void resetRate(double r){_rate = r; };
52          void resetOwes(){_owes = 0;};
53
54      private:
55          double _max_loan;
56          double _rate;
57          double _owes;
58  };
59
60
61  #endif // ACCOUNTABC_H
```

Politechnika Wrocławska

Dziedziczenie

main.cpp | AccountABC.h | **AccountABC.cpp**

```
1   #include <iostream>
2   #include "AccountABC.h"
3
4   using std::cout;
5   using std::ios_base;
6   using std::endl;
7
8   using std::string;
9
10  AccountABC::AccountABC(const string& s, long an, double bal)
11      : _full_name(s)
12      , _account_number(an)
13      , _balance(bal)
14  {}
15
16  void AccountABC::deposit(double amount){
17      if( amount < 0 )
18          cout << "Nie mozna wplacic ujemnej kwoty; "
19               << "wplata anulowana." << endl;
20      else
21          _balance += amount;
22  }
23
24  void AccountABC::withdraw(double amount){
25      _balance -= amount;
26  }
27
```

Politechnika Wrocławska

Dziedziczenie

main.cpp | AccountABC.h | **AccountABC.cpp**

```
28  AccountABC::Formatting AccountABC::setFormat() const{
29      Formatting f;
30      f.flag = cout.setf(ios_base::fixed, ios_base::floatfield);
31      f.pr = cout.precision(2);
32      return f;
33  }
34
35  void AccountABC::restore(Formatting& f) const{
36      cout.setf(f.flag, ios_base::floatfield);
37      cout.precision(f.pr);
38  }
39
```

Politechnika Wrocławska

Dziedziczenie

main.cpp | AccountABC.h | **AccountABC.cpp**

```
40  void Account::withdraw(double amount){
41      Formatting f = setFormat();
42
43      if( amount < 0 )
44          cout << "Nie mozna wyplacic ujemnej kwoty; "
45               << "wyplata anulowana." << endl;
46      else if( amount < balance() )
47          AccountABC::withdraw(amount);
48      else
49          cout << "Zadana suma " << amount
50               << " PLN przekracza dostepne srodki." << endl
51               << "Wyplata anulowana." << endl;
52
53       restore(f);
54  }
55
56  void Account::viewAccount() const{
57      Formatting f = setFormat();
58      cout << "Klient: " << fullName() << endl;
59      cout << "Numer rachunku: " << accountNumber() << endl;
60      cout << "Stan konta: " << balance() << " PLN" << endl;
61      restore(f);
62  }
63
```

Politechnika Wrocławska

Dziedziczenie

main.cpp | AccountABC.h | **AccountABC.cpp**

```
64  AccountPlus::AccountPlus(const string& s, long an, double bal,
65                           double ml, double r)
66      : AccountABC(s, an, bal)
67      , _max_loan(ml)
68      , _rate(r)
69      , _owes(0)
70  {}
71
72  AccountPlus::AccountPlus(const Account & a, double ml, double r)
73      : AccountABC(a)
74      , _max_loan(ml)
75      , _rate(r)
76      , _owes(0)
77  {}
78
```

Politechnika Wrocławska

Dziedziczenie

main.cpp | AccountABC.h | **AccountABC.cpp**

```
79  void AccountPlus::withdraw(double amount){
80      Formatting f = setFormat();
81
82      double bal = balance();
83      if( amount < bal )
84          AccountABC::withdraw(amount);
85      else if( amount < bal + _max_loan - _owes ){
86          double advance = amount - bal;
87          _owes += advance * (1.0 + _rate);
88          cout << "Zadluzenie faktyczne: " << advance << " PLN" << endl;
89          cout << "Odsetki: " << advance * _rate << " PLN" << endl;
90          deposit(advance);
91          AccountABC::withdraw(amount);
92      }
93      else
94          cout << "Przekroczony limit debetu. Operacja anulowana" << endl;
95
96      restore(f);
97  }
98
99  void AccountPlus::viewAccount() const{
100     Formatting f = setFormat();
101     cout << "Klient: " << fullName() << endl;
102     cout << "Numer rachunku: " << accountNumber() << endl;
103     cout << "Stan konta: " << balance() << " PLN" << endl;
104     cout << "Limit debetu: " << _max_loan << endl;
105     cout << "Kwota zadluzenia: " << _owes << endl;
106     cout.precision(3);
107     cout << "Stopa oprocentowania: " << 100*_rate << "%" << endl;
108     restore(f);
109 }
110
```

Politechnika Wrocławska

Podsumowanie

- Przeciążanie operatorów
- Funkcje zaprzyjaźnione
- Dziedziczenie