# Zaawansowane programowanie komputerowe

## Symulacja – podstawy dziedziczenia

Dorota Celińska-Kopczyńska

Uniwersytet Warszawski

Zajęcia
31 marca 2025

# Programowanie obiektowe

- Na programowanie obiektowe można patrzeć jako na symulowanie rozważanego świata:
  - agenci, do których wysyła się komunikaty
  - zobowiązani do realizacji pewnych działań
  - realizujący je wg pewnych metod postępowania. Te metody mogą być ukryte przez zlecającym wykonanie zadania
- Poza komórkami pamięci mamy obiekty (agentów), komunikaty i zobowiązania

# Obiekty i klasy

- Obiekt ma swój własny stan (wartości pewnych zmiennych) i swoje własne zachowanie (operacje)
- Każdy obiekt jest egzemplarzem pewnej klasy – ona określa jego zachowanie
- Zachowanie możemy obserwować wysyłając komunikat – w odpowiedzi obiekt wykona swoją operację
- Klasy to typy danych, które pozwalają na zgromaczenie w jednej zmiennej (obiekcie) zarówno danych, jak i operacji z nimi związanych

# Dziedziczenie

- Jeden z naistotniejszych elementów obiektowości
- Dziedziczenie umożliwia tworzenie hierarchii klas
- Klasy odpowiadają pojęciom występującym w rozważanym świecie. Hierarchie klas pozwalają tworzyć hierarchie pojęć, wyrażając zależności między pojęciami
- Klasa pochodna dziedziczy po klasie bazowej. Klasę pochodną (podklasę) tworzymy wówczas, gdy chcemy opisać bardziej wyspecjalizowane obiekty klasy bazowej (nadklasy)
- Każdy obiekt klasy pochodnej jest też obiektem klasy bazowej

# Zalety dziedziczenia

- Jawne wyrażanie zależności między klasami
- Zachęcanie do ponownego wykorzystania już napisanego kodu
- Unikanie ponownego pisania tych samych fragmentów programu
- Ułatwianie znajdowania potrzebnego kodu
- Możliwość narzucania jednolitego interfejsu grupy spokrewnionych klas

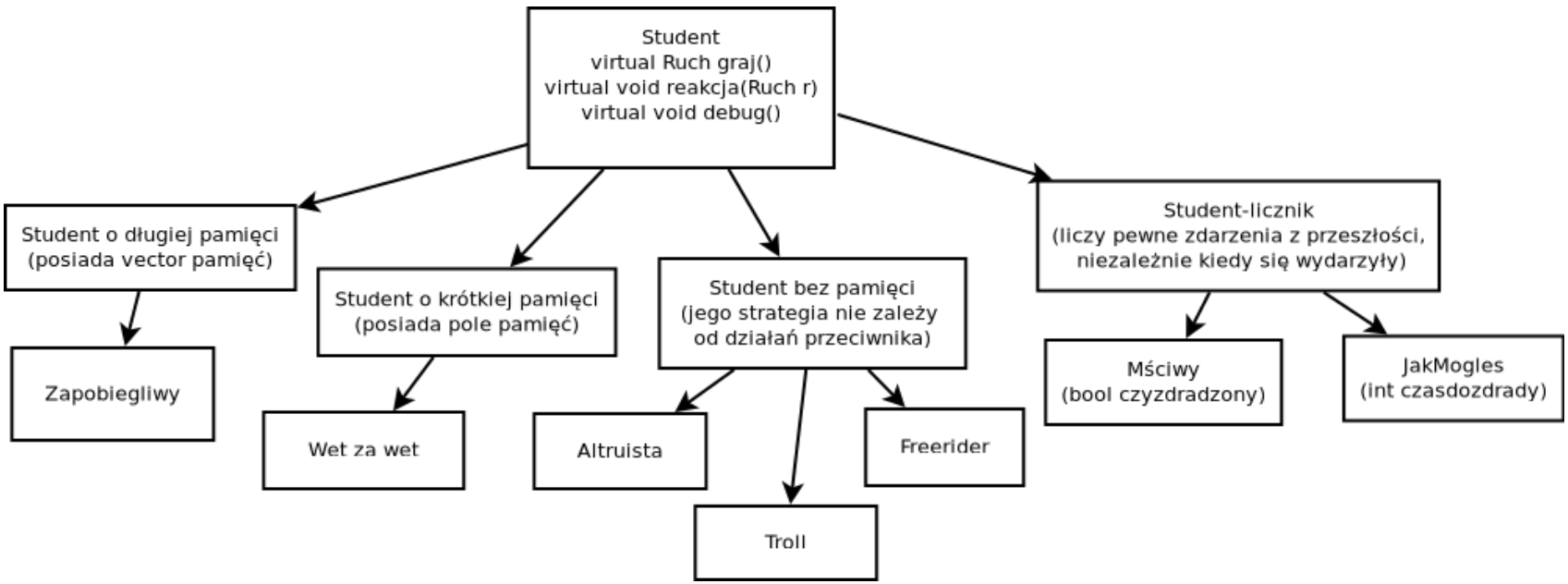
# Typy składowych

- Składowe prywatne – widoczne jedynie w klasie, z której pochodzą i w funkcjach/klasach z nią zaprzyjaźnionych
- Składowe chronione – widoczne w klasie, z której pochodzą, w funkcjach/klasach z nią zaprzyjaźnionych oraz w klasach pochodnych oraz funkcjach/klasach z nimi zaprzyjaźnionych
- Składowe publiczne – widoczne wszędzie tam, gdzie jest widoczna sama klasa

# Zadanie

- Będziemy szykować symulację, w której różne typy Studentów są połączone w zespoły do pracy grupowej
- Każdy student może współpracować lub zdradzić (oszukać) drugiego studenta
- Każdy typ studenta ma inną strategię na pracę w grupie

# Hierarchia


Student
virtual Ruch graj()
virtual void reakcja(Ruch r)
virtual void debug()
Student o długiej pamięci
(posiada vector pamięć)
Zapobiegliwy
Student o krótkiej pamięci
(posiada pole pamięć)
Wet za wet
Student bez pamięci
(jego strategia nie zależy
od działań przeciwnika)
Altruista
Troll
Freerider
Student-licznik
(liczy pewne zdarzenia z przeszłości,
niezależnie kiedy się wydarzyły)
Mściwy
(bool czyzdradzony)
JakMogles
(int czasdozdrady)

# Dziedziczenie – nanoszenie hierarchii do kodu

- Przepisujemy nazwy klas od najbardziej ogólnej do najbardziej szczegółowych
- Klasy pochodne umieszczamy po ich klasach nadrzędnych
- Po nazwie klasy stawiamy dwukropek i podajemy, że dziedziczymy w sposób publiczny (podajemy klasę bezpośrednio nadrzędną w hierarchii)

```
class Student {

};
class Student_o_dlugiej_pamieci: public Student {

};
class Zapobiegliwy: public Student_o_dlugiej_pamieci {

};
```

# Metody klasy nadrzędnej

- Metody mogą być wirtualne (virtual – zostaną doprecyzowane w klasach podrzędnych) lub nie (wtedy działają tak samo dla wszystkich klas podrzędnych)

```
class Student {
  public:
  // metody wirtualne -- zostana doprecyzowane w klasach pochodnych
  virtual Ruch graj() = 0;
  virtual void reakcja(Ruch r) = 0;
  virtual void debug() {};
  virtual ~Student() {};
};
```

# Metody abstrakcyjne

- = 0 – taka metoda musi zostać doprecyzowana dla wszystkich klas pochodnych
- Bez przedefiniowania tej metody nie można stworzyć obiektu danej klasy
- Przykład – metody Ruch graj() nie trzeba byłoby definiować dla "Student licznik", ale Mściwy (którego chcemy stworzyć) musi tę metodę mieć przedefiniowaną

```
class Student {
  public:
  // metody wirtualne -- zostana doprecyzowane w klasach pochodnych
  // =0 -- musza zostac doprecyzowane dla klas pochodnych
  virtual Ruch graj() = 0;
  virtual void reakcja(Ruch r) = 0;
  virtual void debug() {};
  virtual ~Student() {};
};
```

# Jak rozpoznać problem z metodami abstrakcyjnymi?

- Zakomentuję metodę graj() dla klasy Msciwy – kompilator jej nie znajduje

urxvt

```
tehora@archeopteryx ~ λ  cd Documents/dydaktyka/mat_poic++/materialy/zajecia10-11
tehora@archeopteryx ../mat_poic++/materialy/zajecia10-11 λ  g++ dylemat.cpp -fsanitize=address -o dyl
dylemat.cpp: In function ‘int main()’:
dylemat.cpp:119:22: error: invalid new-expression of abstract class type ‘Msciwy’
  119 |     Student* b = new Msciwy;
      |                      ^~~~~~
dylemat.cpp:92:7: note:   because the following virtual functions are pure within ‘Msciwy’:
   92 | class Msciwy: public Student_licznik {
      |       ^~~~~~
dylemat.cpp:18:18: note:     ‘virtual Ruch Student::graj()’
   18 |     virtual Ruch graj() = 0;
      |                  ^~~~
tehora@archeopteryx ../mat_poic++/materialy/zajecia10-11 λ
```

# Metoda debug()

- Metoda czysto techniczna, żeby mogli Państwo sprawdzać, czy kod poprawnie działa
- Napisanie getterów dla klasy podrzędnej nie zadziała (gdyby nimi chcieli Państwo sprawdzać wartości pól publicznych), debug() domyślnie nie ma być definiowany (ale może)

```
class Student {
  public:
  virtual Ruch graj() = 0;
  virtual void reakcja(Ruch r) = 0;
  // metoda ponizej domyslnie nic nie robi, definiowanie jest opcjonalne
  // przydaje sie do sprawdzenia poprawnosci dzialania kodu
  virtual void debug() {};
  virtual ~Student() {};
};
```

# Wirtualny destruktor

- Nie musimy definiować własnych destruktorów dla klas pochodnych, domyślne wystarczą.
- Ale musimy zdefiniować wirtualny destruktor
- Jak destruktor klasy nadrzędnej nie jest wirtualny, usunie się obiekt klasy nadrzędnej (tutaj Student) a nie obiekt właściwej klasy (np. Altruista)

```
class Student {
  public:
  virtual Ruch graj() = 0;
  virtual void reakcja(Ruch r) = 0;
  virtual void debug() {};
  // bez wirtualnego destruktora obiekty beda sie blednie usuwac
  virtual ~Student() {};
};
```

# Wirtualny destruktor

- Tutaj zdefiniowany wirtualny destruktor – wszystko działa

```
urxvt
tehora@archeopteryx ../mat_poic++/materialy/zajecia10-11 λ g++ dylemat.cpp -fsanitize=address -o dyl
tehora@archeopteryx ../mat_poic++/materialy/zajecia10-11 λ ./dyl
0
1
100
99
tehora@archeopteryx ../mat_poic++/materialy/zajecia10-11 λ
```

- Tutaj brak wirtualnego destruktora – wycieki pamięci

```
urxvt
tehora@archeopteryx ../mat_poic++/materialy/zajecia10-11 λ g++ dylemat.cpp -fsanitize=address -o dyl
tehora@archeopteryx ../mat_poic++/materialy/zajecia10-11 λ ./dyl
0
1
100
99
=================================================================
==1873==ERROR: AddressSanitizer: new-delete-type-mismatch on 0x602000000030 in thread T0:
  object passed to delete has wrong type:
  size of the allocated type:   16 bytes;
  size of the deallocated type: 8 bytes.
    #0 0x7f3559d16d69 in operator delete(void*, unsigned long) /build/gcc/src/gcc/libsanitizer/asan/asan_new_delete.cpp:172
    #1 0x5630bf2ec5df in main (/home/tehora/Documents/dydaktyka/mat_poic++/materialy/zajecia10-11/dyl+0x25df)
    #2 0x7f3559743b24 in __libc_start_main (/usr/lib/libc.so.6+0x27b24)
    #3 0x5630bf2ec20d in _start (/home/tehora/Documents/dydaktyka/mat_poic++/materialy/zajecia10-11/dyl+0x220d)

0x602000000030 is located 0 bytes inside of 16-byte region [0x602000000030,0x602000000040)
allocated by thread T0 here:
    #0 0x7f3559d15ca1 in operator new(unsigned long) /build/gcc/src/gcc/libsanitizer/asan/asan_new_delete.cpp:99
    #1 0x5630bf2ec358 in main (/home/tehora/Documents/dydaktyka/mat_poic++/materialy/zajecia10-11/dyl+0x2358)
    #2 0x7f3559743b24 in __libc_start_main (/usr/lib/libc.so.6+0x27b24)

SUMMARY: AddressSanitizer: new-delete-type-mismatch /build/gcc/src/gcc/libsanitizer/asan/asan_new_delete.cpp:172 in operator delete(void*, unsigned long)
==1873==HINT: if you don't care about these errors you may set ASAN_OPTIONS=new_delete_type_mismatch=0
==1873==ABORTING
tehora@archeopteryx ../mat_poic++/materialy/zajecia10-11 λ                                                                                    1
```

# Kopiowania

- Kwestia kopiowań przy dziedziczeniu w C++ jest dosyć skomplikowana
- Tutaj na razie pominiemy ten aspekt – wystarczy nam domyślna implementacja reguły trzech
- *Przypomnienie*: Reguła trzech – jeśli czujemy się w obowiązku zdefiniować samodzielnie którykolwiek spośród trójki: destruktor, konstruktor kopiujący, kopiujący operator przypisania, powinniśmy odnieść się do pozostałych dwóch

# Widoczność pól

- Metody i pola `private` z klasy nadrzędnej nie będą mogły być wykorzystane przez obiekty klasy podrzędnej
- Żeby umożliwić klasom podrzędnym posiadanie wspólnego pola (np. w naszym wypadku typu pamięci) deklarujemy je jako `protected`
- Przy definiowaniu nowych pól domyślne konstruktory mogą już nie wystarczyć

```
class Student_o_dlugiej_pamieci: public Student {
  protected:
  // wszystkie klasy, dla ktorych ta klasa jest nadrzedna beda mogly
  // korzystac z tego pola
    vector<Ruch> pamiec;
  public:
    Student_o_dlugiej_pamieci() = default;
    void reakcja(Ruch r) override {pamiec.push_back(r);};
};
```

# Override

- Gdy doprecyzowujemy metodę w klasie podrzędnej warto oznaczyć ją jako `override` (to nie jest obowiązkowe)
- Gdybyśmy popełnili literówkę/błąd w nagłówku takiej metody, kompilator bardzo łatwo to znajdzie

```
class Student_o_dlugiej_pamieci: public Student {
  protected:
    vector<Ruch> pamiec;
  public:
    Student_o_dlugiej_pamieci() = default;
    // Reakcja jest taka sama dla studentow o dlugiej pamieci
    void reakcja(Ruch r) override {pamiec.push_back(r);};
};
```

# Override

- Zrezygnuję z `override` w klasie Msciwy (nic złego)

```
urxvt
tehora@archeopteryx ../mat_poic++/materialy/zajecia10-11 λ g++ dylemat.cpp -fsanitize=address -o dyl
tehora@archeopteryx ../mat_poic++/materialy/zajecia10-11 λ ./dyl
0
1
100
99
tehora@archeopteryx ../mat_poic++/materialy/zajecia10-11 λ
```

- Zrezygnuję z `override` i zrobię literówkę Graj() – kompilator nie rozpoznał metody

```
urxvt
tehora@archeopteryx ../mat_poic++/materialy/zajecia10-11 λ g++ dylemat.cpp -fsanitize=address -o dyl
dylemat.cpp: In function 'int main()':
dylemat.cpp:119:22: error: invalid new-expression of abstract class type 'Msciwy'
  119 |     Student* b = new Msciwy;
      |                      ^~~~~~
dylemat.cpp:92:7: note:   because the following virtual functions are pure within 'Msciwy':
   92 | class Msciwy: public Student_licznik {
      |       ^~~~~~
dylemat.cpp:18:18: note:     'virtual Ruch Student::graj()'
   18 |     virtual Ruch graj() = 0;
      |                  ^~~~
tehora@archeopteryx ../mat_poic++/materialy/zajecia10-11 λ
```

- Jest `override`, zrobię literówkę Graj() – kompilator pyta

```
urxvt
tehora@archeopteryx ../mat_poic++/materialy/zajecia10-11 λ g++ dylemat.cpp -fsanitize=address -o dyl
dylemat.cpp:98:10: error: 'Ruch Msciwy::Graj()' marked 'override', but does not override
   98 |     Ruch Graj() override {return w;};
      |          ^~~~
dylemat.cpp: In function 'int main()':
dylemat.cpp:119:22: error: invalid new-expression of abstract class type 'Msciwy'
  119 |     Student* b = new Msciwy;
      |                      ^~~~~~
dylemat.cpp:92:7: note:   because the following virtual functions are pure within 'Msciwy':
   92 | class Msciwy: public Student_licznik {
      |       ^~~~~~
dylemat.cpp:18:18: note:     'virtual Ruch Student::graj()'
   18 |     virtual Ruch graj() = 0;
      |                  ^~~~
tehora@archeopteryx ../mat_poic++/materialy/zajecia10-11 λ
```

# Jak tworzyć (i kasować) obiekty?

- Obiekty klasy pochodnej tworzymy z wykorzystaniem new, które przypisuje na wskaźnik do klasy bazowej
- Nawet przy obecności wirtualnego destrutora wskaźnik do klasy bazowej musimy jeszcze usunąć

```
Student* a = new Altruista;
a->graj();
delete a
```

# Sprzątanie

- Tutaj i wirtualny destruktor, i skasowanie wskaźnika

```
urxvt
tehora@archeopteryx ../mat_poic++/materialy/zajecia10-11 λ g++ dylemat.cpp -fsanitize=address -o dyl
tehora@archeopteryx ../mat_poic++/materialy/zajecia10-11 λ ./dyl
0
1
100
99
tehora@archeopteryx ../mat_poic++/materialy/zajecia10-11 λ
```

- Tutaj brak skasowania wskaźnika – wycieki pamięci

```
urxvt
tehora@archeopteryx ../mat_poic++/materialy/zajecia10-11 λ g++ dylemat.cpp -fsanitize=address -o dyl
tehora@archeopteryx ../mat_poic++/materialy/zajecia10-11 λ ./dyl
0
1
100
99

=================================================================
==1967==ERROR: LeakSanitizer: detected memory leaks

Direct leak of 32 byte(s) in 1 object(s) allocated from:
    #0 0x7fed09ca2ca1 in operator new(unsigned long) /build/gcc/src/gcc/libsanitizer/asan/asan_new_delete.cpp:99
    #1 0x5564810a047e in main (/home/tehora/Documents/dydaktyka/mat_poic++/materialy/zajecia10-11/dyl+0x247e)
    #2 0x7fed096d0b24 in __libc_start_main (/usr/lib/libc.so.6+0x27b24)

Direct leak of 16 byte(s) in 1 object(s) allocated from:
    #0 0x7fed09ca2ca1 in operator new(unsigned long) /build/gcc/src/gcc/libsanitizer/asan/asan_new_delete.cpp:99
    #1 0x5564810a0497 in main (/home/tehora/Documents/dydaktyka/mat_poic++/materialy/zajecia10-11/dyl+0x2497)
    #2 0x7fed096d0b24 in __libc_start_main (/usr/lib/libc.so.6+0x27b24)

Direct leak of 16 byte(s) in 1 object(s) allocated from:
    #0 0x7fed09ca2ca1 in operator new(unsigned long) /build/gcc/src/gcc/libsanitizer/asan/asan_new_delete.cpp:99
    #1 0x5564810a0358 in main (/home/tehora/Documents/dydaktyka/mat_poic++/materialy/zajecia10-11/dyl+0x2358)
    #2 0x7fed096d0b24 in __libc_start_main (/usr/lib/libc.so.6+0x27b24)

Direct leak of 8 byte(s) in 1 object(s) allocated from:
    #0 0x7fed09ca2ca1 in operator new(unsigned long) /build/gcc/src/gcc/libsanitizer/asan/asan_new_delete.cpp:99
    #1 0x5564810a02eb in main (/home/tehora/Documents/dydaktyka/mat_poic++/materialy/zajecia10-11/dyl+0x22eb)
    #2 0x7fed096d0b24 in __libc_start_main (/usr/lib/libc.so.6+0x27b24)

SUMMARY: AddressSanitizer: 72 byte(s) leaked in 4 allocation(s).
tehora@archeopteryx ../mat_poic++/materialy/zajecia10-11 λ
```