



Kierunek: Grafika komputerowa i elementy rzeczywistości mieszanej / Informatyka

Rok akademicki: 2022/2023

Przedmiot: Podstawy programowania zorientowane obiektowo

Semestr: II

Grupa dziekańska: 1

Sprawozdanie z: Laboratorium numer 1

data: 6 kwietnia 2023

Grupa robocza: GrupaRobocza06

Skład grupy:

Osoba 1: Rostyslav Hrenitskyi

Osoba 2: Maksym Szczurko

Osoba 3 : Wojciech Gawlas

Osoba 4:

Aktywności realizowane przez członków grupy roboczej, czyli kto i za co był odpowiedzialny podczas laboratorium/zadania domowego oraz utworzenia sprawozdania (min 3 na osobę):

Osoba 1: Rostyslav Hrenitskyi

Zadanie 0, 4, 5.

Osoba 2: Maksym Szczurko

Zadanie 1, 2, 3.

Osoba 3: Wojciech Gawlas

Zadanie 0, 2, 3.

Osoba 4:

Sposoby i narzędzia komunikacji grupy roboczej:

GitHub.

Zadanie 0

Napisać program umożliwiający przechowywanie danych o dużej ilości punktów w przestrzeni 2D. Zastosować podejście strukturalne i obiektowe. Omówić wady i zalety obu rozwiązań. Jaka jest największa różnica między podejściem strukturalnym a obiektowym? Czym jest klasa, metoda i obiekt (instancja klasy)? Jak ma się to do danych i funkcji obsługujących te dane? Czy programowanie obiektowe jest najlepszym rozwiązaniem w każdej sytuacji?

Omówić porządek w kodzie dzięki zamknięciu danych w obiektach.

Zagadka: mamy 100 obiektów, każdy po 100 bajtów cech i 400 bajtów kodu metod. Ile bajtów trzeba w pamięci na przechowanie tych stu obiektów?

Wyjaśnić kwestię niezależnych danych (cech) w każdym obiekcie oraz istotę jednej funkcji (metody) dla wszystkich obiektów utworzonych na bazie tej samej klasy.

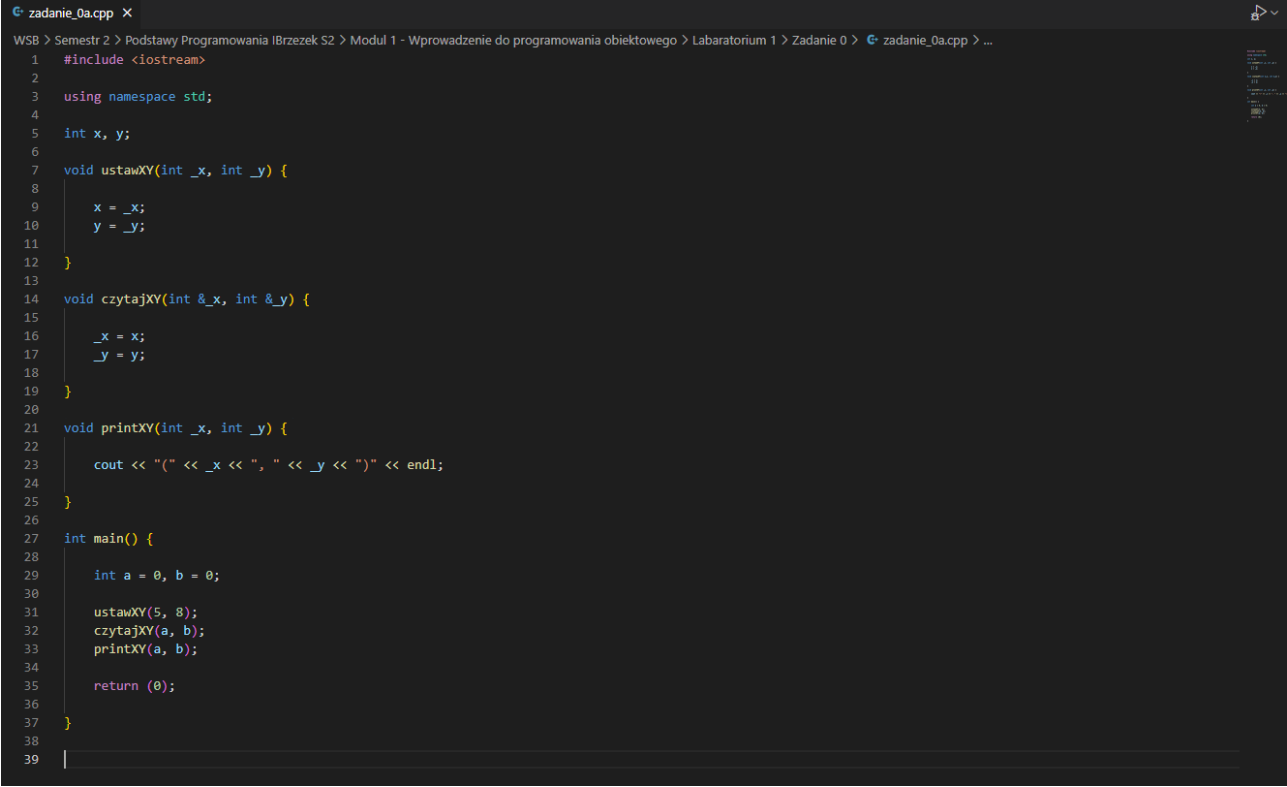
Sprawdzić ile pamięci zajmuje obiekt.

Omówić metodykę tworzenia identyfikatorów w stylu camelCase oraz snake_case.

Rozwiązanie zadania 0

Napisać program umożliwiający przechowywanie danych o dużej ilości punktów w przestrzeni 2D. Zastosować podejście strukturalne i obiektowe.

Podejście strukturalne:



```
zadanie_0a.cpp X
WSB > Semestr 2 > Podstawy Programowania IBrzezek S2 > Moduł 1 - Wprowadzenie do programowania obiektowego > Laboratorium 1 > Zadanie 0 > zadanie_0a.cpp > ...
1  #include <iostream>
2
3  using namespace std;
4
5  int x, y;
6
7  void ustawXY(int _x, int _y) {
8
9      x = _x;
10     y = _y;
11
12 }
13
14 void czytajXY(int &x, int &y) {
15
16     _x = x;
17     _y = y;
18
19 }
20
21 void printXY(int _x, int _y) {
22
23     cout << "(" << _x << ", " << _y << ")" << endl;
24
25 }
26
27 int main() {
28
29     int a = 0, b = 0;
30
31     ustawXY(5, 8);
32     czytajXY(a, b);
33     printXY(a, b);
34
35     return (0);
36
37 }
38
39
```

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
PS C:\Users\R\Desktop\Code\CPP> & 'c:\Users\R\.vscode\extensions\ms-vscode.cpptools-1.14.4-win32-x64\debugAdapters\bin\WindowsDebugLauncher.exe' '--stdin=Microsoft-MIEngine-In-ku
r3tfe1.14m' '--stdout=Microsoft-MIEngine-Out-czn3ebwj.ujz' '--stderr=Microsoft-MIEngine-Error-odbmggru.4uj' '--pid=Microsoft-MIEngine-Pid-ryuhcbbp.gte' '--dbgExe=C:\msys64\mingw64
\bin\gdb.exe' '--interpreter=mi'
(5, 8)
PS C:\Users\R\Desktop\Code\CPP> []
```

source: ...\\Modul 1 - Wprowadzenie do programowania obiektowego\\Laboratorium 1\\Zadanie 0\\zadanie_0a.cpp

Podjęście obiektowe:

```
zadanie_0b.cpp X
WSB > Semestr 2 > Podstawy Programowania IBrzezek S2 > Modul 1 - Wprowadzenie do programowania obiektowego > Laboratorium 1 > Zadanie 0 > zadanie_0b.cpp > ...
1  #include <iostream>
2
3  using namespace std;
4
5  class Punkt {
6  private:
7      int x, y;
8      string nazwa;
9
10 public:
11     void ustawXY(int _x, int _y) {
12         x = _x;
13         y = _y;
14     }
15
16     void czytajXY(int &x, int &y) {
17         _x = x;
18         _y = y;
19     }
20
21     void printXY() {
22         cout << "(" << x << ", " << y << ")" << endl;
23     }
24 };
25
26 int main() {
27     int a, b;
28     Punkt A, B;
29
30     A.ustawXY(10, 20);
31     A.czytajXY(a, b);
32     A.printXY();
33
34     return (0);
35 }
36
37
38
39
40
41
42
43
44
45
46
```

```
PS C:\Users\R\Desktop\Code\CPP> & 'c:\Users\R\.vscode\extensions\ms-vscode.cpptools-1.14.4-win32-x64\debugAdapters\bin\WindowsDebugLauncher.exe' '--stdin=Microsoft-MIEngine-In-3c
wmn2h.vwl' '--stdout=Microsoft-MIEngine-Out-txizfpva.i13' '--stderr=Microsoft-MIEngine-Error-siagivkh.spy' '--pid=Microsoft-MIEngine-Pid-cnygrg3a.job' '--dbgExe=C:\msys64\mingw64
\bin\gdb.exe' '--interpreter=mi'
(10, 20)
PS C:\Users\R\Desktop\Code\CPP> []
```

source: ...\\Modul 1 - Wprowadzenie do programowania obiektowego\\Laboratorium 1\\Zadanie 0\\zadanie_0b.cpp

Omówić wady i zalety obu rozwiązań.

Podejście strukturalne:

- **Zalety:**
 - Łatwe do zrozumienia dla początkujących programistów.
 - Prostsze i bardziej zwyczajne podejście, co może przyspieszyć rozwój projektu, zwłaszcza jeśli nie ma potrzeby tworzenia zaawansowanej logiki.
- **Wady:**
 - Podejście jest słabe pod względem hermetyzacji. Niema sposoby na ukrycie danych przed użytkownikiem. Nie ma wewnętrznych metod.
 - Trudno aktualizować kod.
 - Trudniej tworzyć skomplikowane projekty.

Podejście obiektowe:

- **Zalety:**
 - Kod jest modularny. Klasa może zawierać wiele metod, dla korzystania zewnętrzne.
 - Łatwe zarządzanie kodem. Kod jest uporządkowany.
 - Kod jest bardziej czytelny.
 - Łatwiejsze tworzenie bardziej skomplikowanych projektów.
- **Wady:**
 - Może być trudniejsze dla zrozumienia dla początkujących programistów.
 - Projektowanie klas wymaga więcej czasu.
 - Kod może być bardziej wymagający pod względem wydajności.

Jaka jest największa różnica między podejściem strukturalnym a obiektowym?

Największą różnicą między podejściem strukturalnym a obiektowym jest sposób organizacji kodu i danych. W podejściu strukturalnym program skupia się na operacjach wykonywanych na danych, podczas gdy w podejściu obiektowym program skupia się na danych i ich zachowaniu.

Czym jest klasa, metoda i obiekt (instancja klasy)?

W języku programowania obiektowego, klasa, metoda i obiekt to trzy podstawowe pojęcia, które są kluczowe dla tworzenia i wykorzystywania programów.

Klasa to szablon lub definicja, która opisuje zestaw atrybutów (czyli właściwości lub zmienne) i metod (czyli funkcje), które będą dostępne dla obiektów, które będą oparte na tej klasie.

Metoda to funkcja zdefiniowana wewnątrz klasy, która wykonuje określone działania na atrybutach klasy i może zwracać wynik. Metoda jest sposobem, aby wykonywać określone zadania lub operacje na danych zawartych w klasie.

Obiekt to instancja klasy. W przypadku, gdy klasa jest szablonem, który definiuje zachowanie pewnej kategorii obiektów, to obiekt jest konkretną reprezentacją takiego obiektu.

Jak ma się to do danych i funkcji obsługujących te dane?

Klasy, metody i obiekty są często używane do organizowania i manipulowania danymi w programowaniu obiektowym, podczas gdy funkcje są ogólnymi blokami kodu, które mogą działać na danych, bez konieczności tworzenia klasy czy obiektu.

Czy programowanie obiektowe jest najlepszym rozwiązaniem w każdej sytuacji?

Programowanie obiektowe jest lepszym rozwiązaniem w bardziej skomplikowanych projektach gdzie pracują z większą ilością danych.

Zagadka: mamy 100 obiektów, każdy po 100 bajtów cech i 400 bajtów kodu metod. Ile bajtów trzeba w pamięci na przechowanie tych stu obiektów?

100 obiektów * 500 bajtów na jeden obiekt = 50 000 bajtów.

Aby przechować 100 obiektów potrzebujemy 50 000 bajtów.

Wyjaśnić kwestię niezależnych danych (cech) w każdym obiekcie oraz istotę jednej funkcji (metody) dla wszystkich obiektów utworzonych na bazie tej samej klasy.

W każdym obiekcie klasy **Punkt** znajdują się trzy prywatne dane: **x**, **y** i **nazwa**. Te dane są niezależne w każdym obiekcie, co oznacza, że każdy obiekt klasy **Punkt** może mieć inne wartości dla tych danych. Metody klasy **Punkt** - **ustawXY()**, **czytajXY()** i **printXY()** są zdefiniowane tylko raz, ale są dostępne dla wszystkich obiektów utworzonych na bazie tej samej klasy.

Sprawdzić ile pamięci zajmuje obiekt.

W tym przykładzie jeden obiekt klasy **Punkt** składa się z dwóch zmiennych typu **int** (każda po 4 bajty) i jednej zmiennej typu **string** (w zależności od długości ciągu znaków). Dla uproszczenia można przyjąć, że długość ciągu to 1 bajt.

Zatem jeden obiekt klasy **Punkt** zajmuje $2 * 4 + 1 = 9$ bajtów. W tym przypadku mamy dwa obiekty klasy **Punkt**. Zatem w sumie oba obiekty zajmują $2 * 9 = 18$ bajtów.

Omówić metodykę tworzenia identyfikatorów w stylu camelCase oraz snake_case.

W stylu **camelCase** pierwsze słowo jest zapisane małą literą, a każde kolejne słowo jest pisane z wielką literą początkową, np. **nazwaPliku**.

W stylu **snake_case** każde słowo w identyfikatorze jest oddzielone znakiem podkreślenia "_", np. **nazwa_pliku**.

Zadanie 1

Utworzyć klasę przechowującą postaci Lego Ninjago. Zawartość:

gatunek (kategoria postaci), imię, obrona, atak, szybkość

Utworzyć kilka obiektów zgodnych z podanymi przykładowymi postaciami.

Zapisać do nich dane, bezpośrednio zapisując cechy do obiektu.

Jak wygląda zapis? Podobny do zapisu do struktury? Kto może zapisywać do obiektu?

Zastanowić się czym klasa różni się w tym wypadku od struktury? Mamy tylko dane, cechy, atrybuty. Jakaś różnica? Po co nam zatem obiekty?

Odczytać cechy bezpośrednio z obiektów.

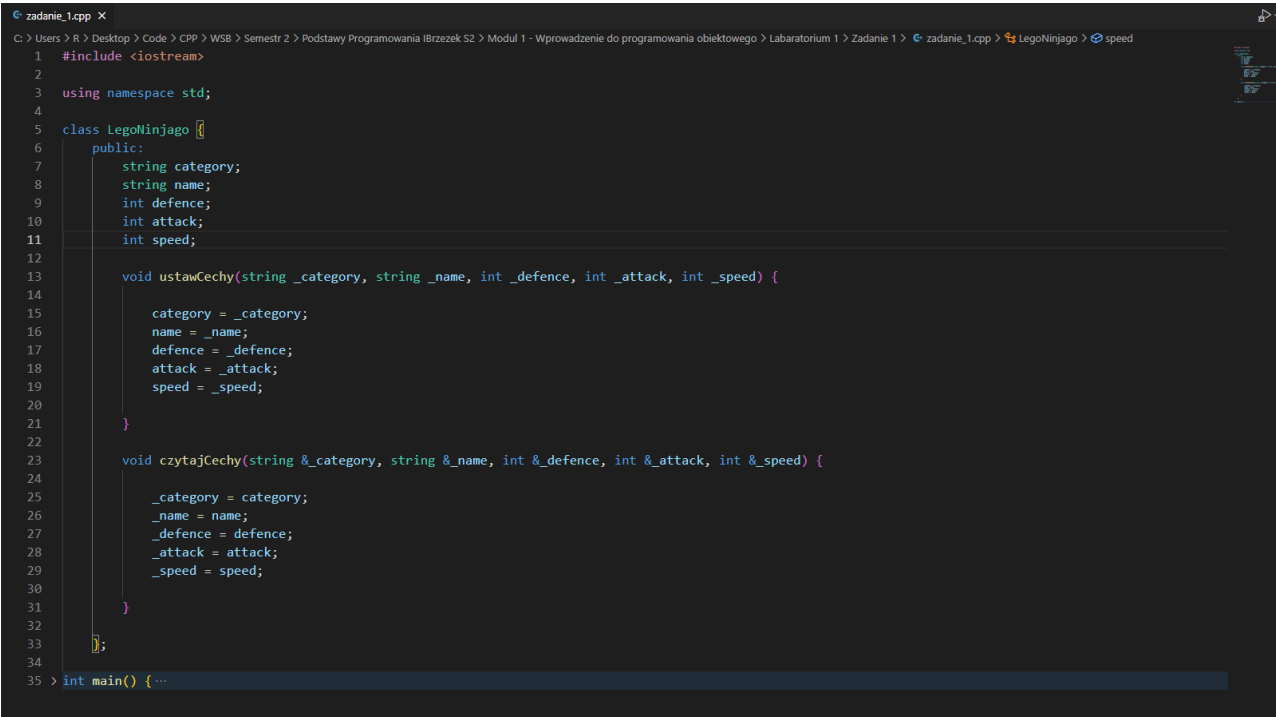
Jak wygląda odczyt? Kto może czytać z obiektu?

Jak tworzone i obsługiwane są zmienne obiektowe tego typu?

Rozwiązanie zadania 1

Utworzyć klasę przechowującą postaci Lego Ninjago. Zawartość:

gatunek (kategoria postaci), imię, obrona, atak, szybkość.



```
zadanie_1.cpp X
C:\Users\R\Desktop > Code > CPP > WSB > Semestr 2 > Podstawy Programowania IBrzemek S2 > Modul 1 - Wprowadzenie do programowania obiektowego > Labaratorium 1 > Zadanie 1 > zadanie_1.cpp > LegoNinjago > speed

1  #include <iostream>
2
3  using namespace std;
4
5  class LegoNinjago {
6  public:
7      string category;
8      string name;
9      int defence;
10     int attack;
11     int speed;
12
13     void ustawCechy(string _category, string _name, int _defence, int _attack, int _speed) {
14
15         category = _category;
16         name = _name;
17         defence = _defence;
18         attack = _attack;
19         speed = _speed;
20     }
21
22     void czytajCechy(string &_category, string &_name, int &_defence, int &_attack, int &_speed) {
23
24         _category = category;
25         _name = name;
26         _defence = defence;
27         _attack = attack;
28         _speed = speed;
29     }
30
31 };
32
33
34
35 > int main() { ...
```

source: ...\\Modul 1 - Wprowadzenie do programowania obiektowego\\Labaratorium 1\\Zadanie 1\\zadanie_1.cpp

Utworzyć kilka obiektów zgodnych z podanymi przykładowymi postaciami.
Zapisać do nich dane, bezpośrednio zapisując cechy do obiektu.

```
zadanie_1.cpp
C:\Users\R\Desktop> Code > CPP > WSB > Semestr 2 > Podstawy Programowania IBrzezek S2 > Modul 1 - Wprowadzenie do programowania obiektowego > Labaratorium 1 > Zadanie 1 > zadanie_1.cpp > main()
1  #include <iostream>
2
3  using namespace std;
4
5  class LegoNinjago {
34
35  int main() {
36
37      string a; string b; int c; int d; int e;
38
39      LegoNinjago Zugu;
40      Zugu.category = "Wojownik";
41      Zugu.name = "Zugu";
42      Zugu.defence = 43;
43      Zugu.attack = 76;
44      Zugu.speed = 34;
45
46      LegoNinjago Kai;
47      Kai.category = "Wojownik";
48      Kai.name = "Kai";
49      Kai.defence = 58;
50      Kai.attack = 71;
51      Kai.speed = 47;
52
53      LegoNinjago Yang;
54      Yang.category = "Czarownik";
55      Yang.name = "Yang";
56      Yang.defence = 65;
57      Yang.attack = 100;
58      Yang.speed = 65;
59
60      Zugu.czytajCechy(a, b, c, d, e);
61
62      cout << "Gatunek: " << a << ". " << endl << "Imie: " << b << ". " << endl << "Obrona: " << c << ". " << endl;
63      cout << "Ataka: " << d << ". " << endl << "Predkosc: " << e << ". " << endl;
64
65      return (0);
66
67  }
```

source: ...\\Modul 1 - Wprowadzenie do programowania obiektowego\\Labaratorium 1\\Zadanie 1\\zadanie_1.cpp

Jak wygląda zapis? Podobny do zapisu do struktury? Kto może zapisywać do obiektu?

Ten kod przedstawia prosty przykład klasy **LegoNinjago** z pięcioma polami (category, name, defence, attack, speed) oraz dwoma metodami (**ustawCechy** i **czytajCechy**). Klasa **LegoNinjago** jest podobna do struktury - obie konstrukcje pozwalają na grupowanie powiązanych ze sobą zmiennych.

Zastanowić się czym klasa różni się w tym wypadku od struktury? Mamy tylko dane, cechy, atrybuty. Jakaś różnica? Po co nam zatem obiekty?

Różnicą między klasą a strukturą jest to, że klasa ma domyślnie ustawiony dostęp prywatny do swoich pól i metod, podczas gdy struktura ma domyślnie dostęp publiczny. Obiekty można ich do przechowywania i przetwarzania danych związanych z klasą.

Odczytać cechy bezpośrednio z obiektów. Jak wygląda odczyt? Kto może czytać z obiektu?

W tym konkretnym przypadku, pola są publiczne. Więc można zapisywać do obiektów bezpośrednio.

```
zadanie_1.cpp
C:\Users\R\Desktop> Code > CPP > WSB > Semestr 2 > Podstawy Programowania IBrzemek S2 > Modul 1 - Wprowadzenie do programowania obiektowego > Laboratorium 1 > Zadanie 1 > zadanie_1.cpp > main()
38
39     LegoNinjago Zugu;
40     Zugu.category = "Wojownik";
41     Zugu.name = "Zugu";
42     Zugu.defence = 43;
43     Zugu.attack = 76;
44     Zugu.speed = 34;
45
46     LegoNinjago Kai;
47     Kai.category = "Wojownik";
48     Kai.name = "Kai";
49     Kai.defence = 58;
50     Kai.attack = 71;
51     Kai.speed = 47;
52
53     LegoNinjago Yang;
54     Yang.category = "Czarownik";
55     Yang.name = "Yang";
56     Yang.defence = 65;
57     Yang.attack = 100;
58     Yang.speed = 65;
```

source: ...\\Modul 1 - Wprowadzenie do programowania obiektowego\\Laboratorium 1\\Zadanie 1\\zadanie_1.cpp

Również metoda **UstawCechy** umożliwia zapis do pól.

```
zadanie_1.cpp
C:\Users\R\Desktop> Code > CPP > WSB > Semestr 2 > Podstawy Programowania IBrzemek S2 > Modul 1 - Wprowadzenie do programowania obiektowego > Laboratorium 1 > Zadanie 1 > zadanie_1.cpp > main()
59
60     a = "wojownik", b = abc, c = 65, d = 70, e = 35;
61     Zugu.ustawCechy(a, b, c, d, e);
62
63     cout << "Gatunek: " << a << "." << endl << "Imie: " << b << "." << endl << "Obrona: " << c << "." << endl;
64     cout << "Ataka: " << d << "." << endl << "Predkosc: " << e << "." << endl;
65
66     return (0);
67
68 }
```

```
PS C:\Users\R> & 'c:\Users\R\.vscode\extensions\ms-vscode.cpptools-1.14.5-win32-x64\debugAdapters\bin\WindowsDebugLauncher.exe' '--stdin=Microsoft-MIEngine-In-hzoniixh.uv' '--stdout=Microsoft-MIEngine-Out-4534u5en.bxy' '--stderr=Microsoft-MIEngine-Error-txq00ndm.ras' '--pid=Microsoft-MIEngine-Pid-wdk1wonr.wlv' '--dbgExe=C:\msys64\mingw64\bin\gdb.exe' '--interpreter=mi'
Gatunek: Wojownik.
Imie: Zugu.
Obrona: 43.
Ataka: 76.
Predkosc: 34.
PS C:\Users\R>
```

source: ...\\Modul 1 - Wprowadzenie do programowania obiektowego\\Laboratorium 1\\Zadanie 1\\zadanie_1.cpp

Jak tworzone i obsługiwane są zmienne obiektowe tego typu?

Zmienne obiektowe klasy **LegoNinjago** są tworzone poprzez deklarację obiektu z nazwą klasy i nazwą zmiennej, np. **{ LegoNinjago Zugu; }**

Aby odwołać się do pól lub metod obiektu, używa się operatora kropki, np. **{ Zugu.name = "Zugu"; }**

Zadanie 2

Rozbudować powyższy program o metodę wyświetlającą tekst, "Jestem <gatunek>, nazywam się <imie> mam takie moce: obrona=<..> atak=<...>, szybkość=<...>

Metoda niech nazywa się "pokazDane".

Omówić metodykę tworzenia identyfikatorów w stylu camelCase oraz snake_case.

Czy taka metoda ma dostęp do wnętrza klasy?

Utworzyć kolejną metodę wczytującą z klawiatury dane do cech obiektu. Nazwa "wczytajDane"

Utworzyć kilka obiektów i wywołać obie metody na każdym z nich.

Wczytać dane i kolejno wywołać metodę ich wyświetlania.

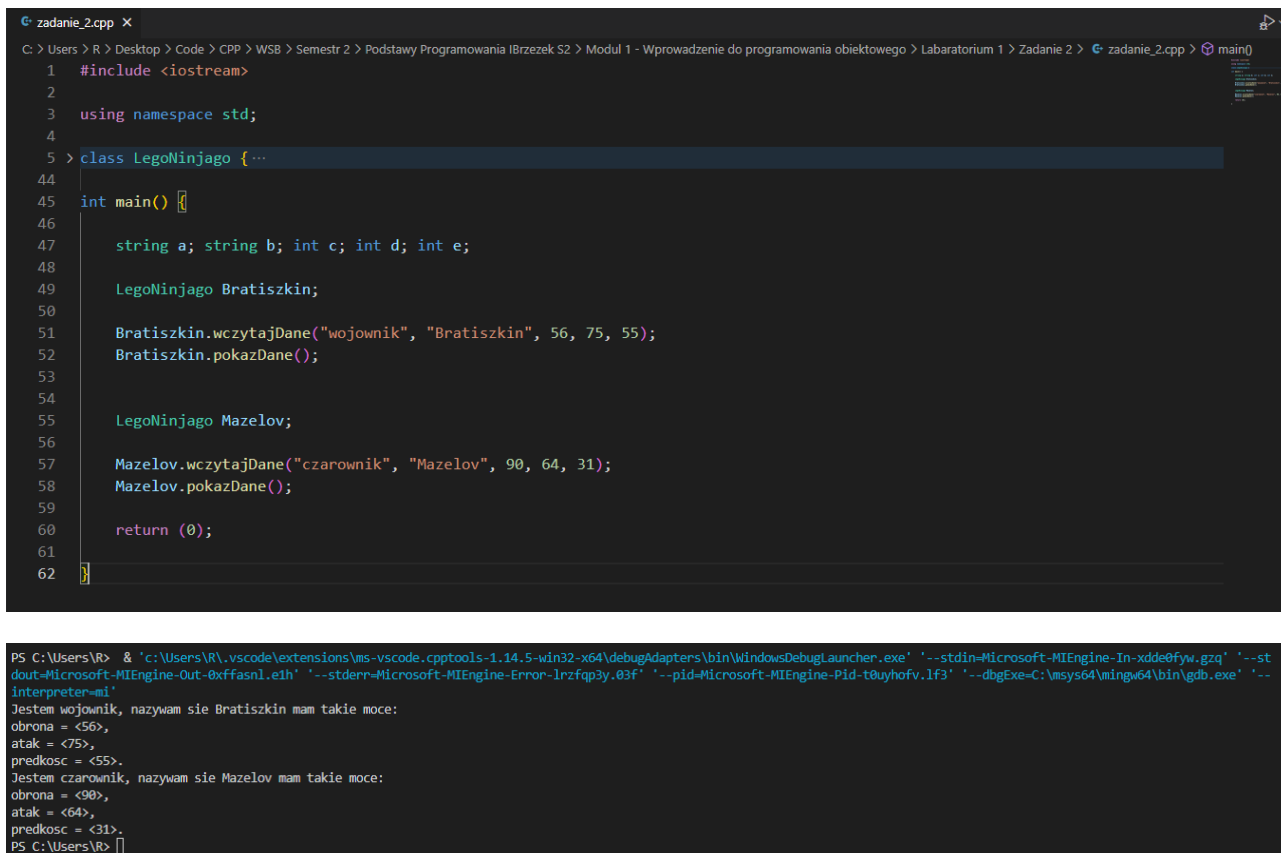
Jak działa taka metoda dodawania danych i odczytu?

Omówić porządek w kodzie dzięki zamknięciu danych w obiektach.

Rozwiązanie zadania 2

Rozbudować powyższy program o metodę wyświetlającą tekst, "Jestem <gatunek>, nazywam się <imie> mam takie moce: obrona=<..> atak=<...>, szybkość=<...>. Metoda niech nazywa się "pokazDane". Utworzyć kolejną metodę wczytującą z klawiatury dane do cech obiektu. Nazwa "wczytajDane". Utworzyć kilka obiektów i wywołać obie metody na każdym z nich. Wczytać dane i kolejno wywołać metodę ich wyświetlania.

```
zadanie_2.cpp X
C:\Users\R\Desktop\Code\CPP\WSB\Semestr 2\Podstawy Programowania IBrzezek S2\Modul 1 - Wprowadzenie do programowania obiektowego\Laboratorium 1\Zadanie 2\zadanie_2.cpp main()
1 #include <iostream>
2
3 using namespace std;
4
5 class LegoNinjabo {
6     private:
7         string category;
8         string name;
9         int defence;
10        int attack;
11        int speed;
12    public:
13        void wczytajDane(string _category, string _name, int _defence, int _attack, int _speed) {
14            category = _category;
15            name = _name;
16            defence = _defence;
17            attack = _attack;
18            speed = _speed;
19        }
20
21        void czytajDane(string &category, string &name, int &defence, int &attack, int &speed) {
22            _category = category;
23            _name = name;
24            _defence = defence;
25            _attack = attack;
26            _speed = speed;
27        }
28
29        void pokazDane() {
30            cout << "Jestem " << category << ", " << "nazywam sie " << name << " mam takie moce: " << endl;
31            cout << "obrona = <" << defence << ">," << endl;
32            cout << "atak = <" << attack << ">," << endl;
33            cout << "predkosc = <" << speed << ">." << endl;
34        }
35    };
36
37 int main() {
```



The image shows a screenshot of a C++ program in a code editor and its execution output in a terminal. The code defines a `LegoNinjabo` class with two methods, `wczytajDane` and `pokazDane`, and uses it in the `main` function to create and display data for two characters: `Bratyszkina` and `Mazelow`.

```
1 #include <iostream>
2
3 using namespace std;
4
5 class LegoNinjabo { ...
44
45 int main() {
46
47     string a; string b; int c; int d; int e;
48
49     LegoNinjabo Bratyszkin;
50
51     Bratyszkin.wczytajDane("wojownik", "Bratyszkin", 56, 75, 55);
52     Bratyszkin.pokazDane();
53
54
55     LegoNinjabo Mazelov;
56
57     Mazelov.wczytajDane("czarownik", "Mazelov", 90, 64, 31);
58     Mazelov.pokazDane();
59
60     return (0);
61
62 }
```

The terminal output shows the program's execution, displaying the character data in a structured format:

```
PS C:\Users\VR> & 'c:\Users\VR\.vscode\extensions\ms-vscode.cpptools-1.14.5-win32-x64\debugAdapters\bin\WindowsDebugLauncher.exe' '--stdin=Microsoft-MIEngine-In-xd0e0fyw.gzq' '--stdout=Microsoft-MIEngine-Out-0x0ffasnl.eih' '--stderr=Microsoft-MIEngine-Error-lr3fq3y.03f' '--pid=Microsoft-MIEngine-Pid-t0uyhofv.1f3' '--dbgExe=C:\msys64\mingw64\bin\gdb.exe' '--interpreter=mi'
Jestem wojownik, nazywam sie Bratyszkin mam takie moce:
obrona = <56>,
atak = <75>,
predkosc = <55>.
Jestem czarownik, nazywam sie Mazelov mam takie moce:
obrona = <90>,
atak = <64>,
predkosc = <31>.
PS C:\Users\VR>
```

source: ...\\Modul 1 - Wprowadzenie do programowania obiektowego\\Labaratorium 1\\Zadanie 2\\zadanie_2.cpp

Omówić metodykę tworzenia identyfikatorów w stylu camelCase oraz snake_case.

W stylu **camelCase** pierwsze słowo jest zapisane małą literą, a każde kolejne słowo jest pisane z wielką literą początkową, np. **nazwaPliku**.

W stylu **snake_case** każde słowo w identyfikatorze jest oddzielone znakiem podkreślenia "_", np. **nazwa_pliku**.

Czy taka metoda ma dostęp do wnętrza klasy?

Tak, metoda **pokazDane** ma dostęp do wnętrza klasy, ponieważ jest ona zdefiniowana jako publiczna w tej klasie.

Jak działa taka metoda dodawania danych i odczytu?

Metoda **wczytajDane** służy do wprowadzania danych do pól prywatnych klasy, a metoda **czytajDane** służy do odczytu tych danych z pól prywatnych.

Omówić porządek w kodzie dzięki zamknięciu danych w obiektach.

Dzięki zamknięciu danych w obiektach możliwe jest łatwe zarządzanie i przetwarzanie danych związanych z klasą, a metody klasy umożliwiają ustawianie i odczytywanie danych dla danego obiektu. Porządek w kodzie jest zachowany dzięki uporządkowaniu danych związanych z klasą w jednym miejscu. Dzięki temu, programista może łatwiej zrozumieć, jakie dane są związane z tą klasą i jakie operacje można na nich wykonywać.

Zadanie 3

Zabezpieczyć cechy obiektu przed możliwością modyfikacji z zewnątrz, pozwolić jedynie metodom na dostęp do cech.

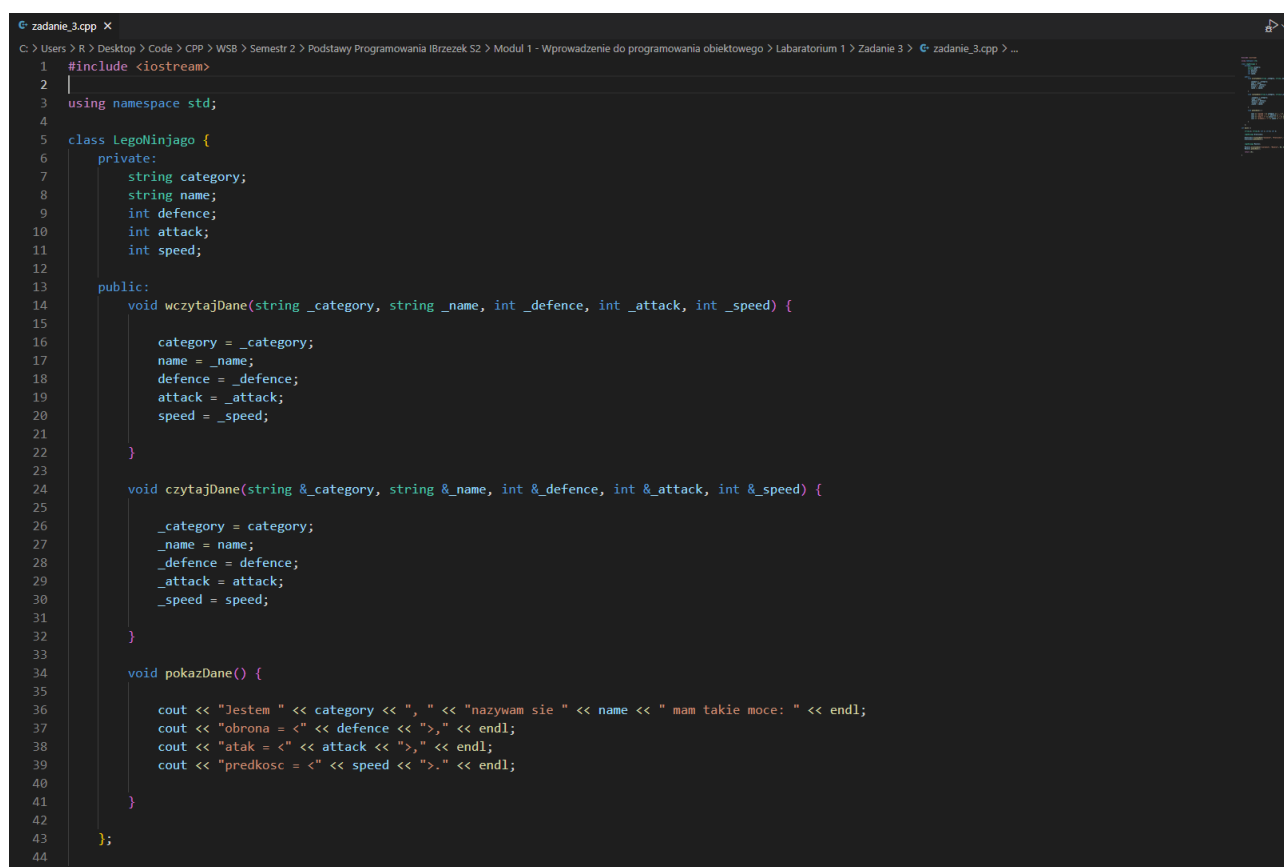
Sprawdzić działanie.

Wyjaśnić kwestię sekcji dostępu do elementów obiektu.

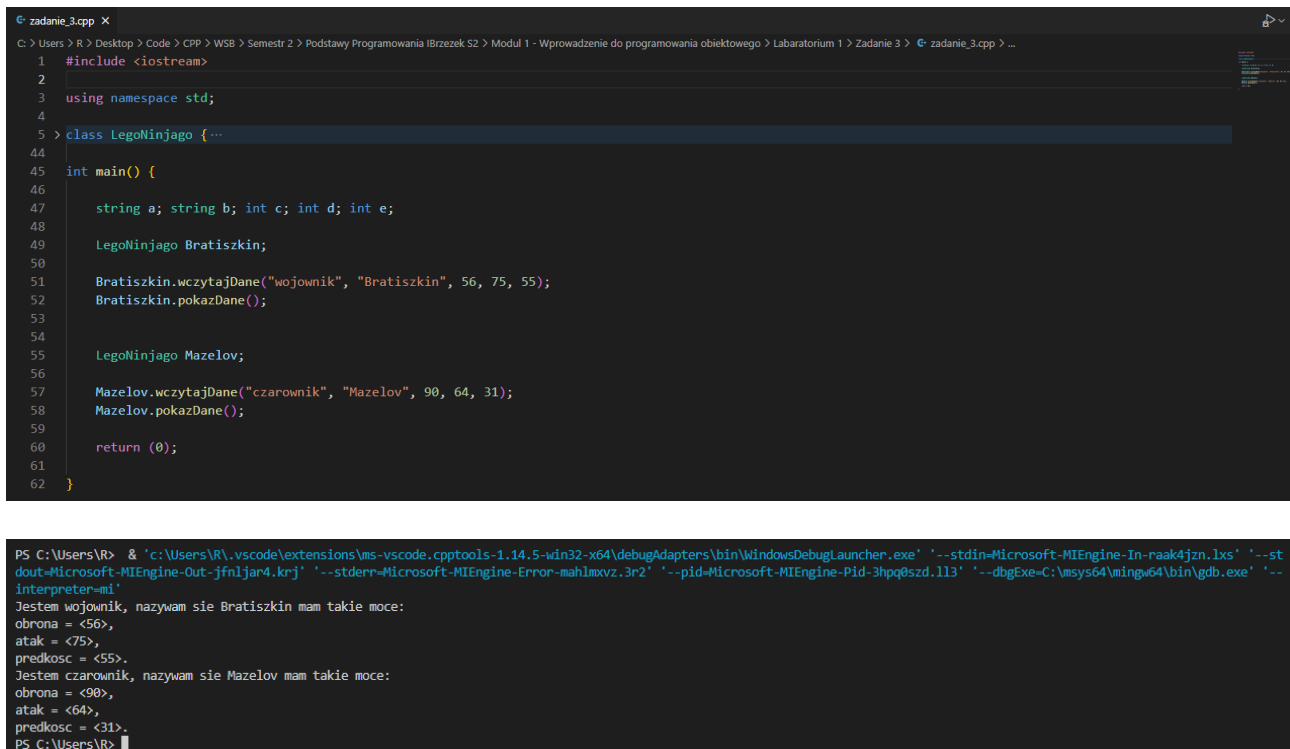
Podać przykłady zastosowania i sens takich funkcjonalności.

Rozwiązanie zadania 3

Zabezpieczyć cechy obiektu przed możliwością modyfikacji z zewnątrz, pozwolić jedynie metodom na dostęp do cech. Sprawdzić działanie.



```
1 #include <iostream>
2
3 using namespace std;
4
5 class LegoNinjabo {
6     private:
7         string category;
8         string name;
9         int defence;
10        int attack;
11        int speed;
12    public:
13        void wczytajDane(string _category, string _name, int _defence, int _attack, int _speed) {
14            category = _category;
15            name = _name;
16            defence = _defence;
17            attack = _attack;
18            speed = _speed;
19        }
20
21        void czytajDane(string &category, string &name, int &defence, int &attack, int &speed) {
22            _category = category;
23            _name = name;
24            _defence = defence;
25            _attack = attack;
26            _speed = speed;
27        }
28
29        void pokazDane() {
30            cout << "Jestem " << category << ", " << "nazywam sie " << name << " mam takie moce: " << endl;
31            cout << "obrona = <" << defence << ">," << endl;
32            cout << "atak = <" << attack << ">," << endl;
33            cout << "predkosc = <" << speed << ">." << endl;
34        }
35    };
36
37 }
```



The image shows a C++ code editor window titled 'zadanie_3.cpp' with the following code:

```
1 #include <iostream>
2
3 using namespace std;
4
5 class LegoNinjabo { ...
44
45 int main() {
46     string a; string b; int c; int d; int e;
47
48     LegoNinjabo Bratiszkin;
49
50     Bratiszkin.wczytajDane("wojownik", "Bratiszkin", 56, 75, 55);
51     Bratiszkin.pokazDane();
52
53
54     LegoNinjabo Mazelov;
55
56     Mazelov.wczytajDane("czarownik", "Mazelov", 90, 64, 31);
57     Mazelov.pokazDane();
58
59
60     return (0);
61 }
62 }
```

Below the code editor is a terminal window showing the output of the program:

```
PS C:\Users\R> & "c:\Users\R\.vscode\extensions\ms-vscode.cpptools-1.14.5-win32-x64\debugAdapters\bin\WindowsDebugLauncher.exe" "--stdin=Microsoft-MIEngine-In-raak4jzn.lxs" "--stdout=Microsoft-MIEngine-Out-jfnljar4.krj" "--stderr=Microsoft-MIEngine-Error-mahlmvz.3r2" "--pid=Microsoft-MIEngine-Pid-3hpq0szd.113" "--dbgExe=C:\msys64\mingw64\bin\gdb.exe" "--interpreter=mi"
Jestem wojownik, nazywam sie Bratiszkin mam takie moce:
obrona = <56>,
atak = <75>,
predkosc = <55>.
Jestem czarownik, nazywam sie Mazelov mam takie moce:
obrona = <90>,
atak = <64>,
predkosc = <31>.
PS C:\Users\R>
```

source: ...\\Modul 1 - Wprowadzenie do programowania obiektowego\\Labaratorium 1\\Zadanie 2\\zadanie_2.cpp

Wyjaśnić kwestię sekcji dostępu do elementów obiektu. Podać przykłady zastosowania i sens takich funkcjonalności.

Sekcja dostępu określa, jakie elementy klasy są dostępne dla innych części programu.

W klasie **LegoNinjabo**:

prywatnymi są:

- **category**
- **name**
- **defence**
- **attack**
- **speed**

publicznymi są:

- metoda **wczytajDane**
- metoda **czytajDane**
- metoda **pokazDane**

Przykłady zastosowania i sens takich funkcjonalności:

- Ukrywanie wewnętrznej implementacji klasy przed innymi częściami programu.
- Zwiększenie bezpieczeństwa programu poprzez kontrolowanie dostępu.
- Zwiększenie elastyczności kodu.

Zadanie 4

Napisać mini grę "MikroNinjago".

Utworzyć grę, gdzie użytkownik wybierze pierwszą i drugą postać i stoczą one "pojedynek" na wybraną swoją moc. Jeśli Atak pierwszej jest większy niż obrona drugiej to wygrywa pierwszy i wspak. Ale jeśli te parametry są identyczne to decyduje Szybkość. Zaprojektować klasy i ich pola (cechy, właściwości). Opracować małe menu i logikę gry. Jak ustalić wartości cech? Można ręcznie lub losowo.

Rozwiązanie zadania 4

Gra MikroNinjago:

1 wariacja(ze stworzeniem swojej postaci):

```
PS C:\Users\R\Desktop\Code\CPP> & 'c:\Users\R\.vscode\extensions\ms-vscode.cpptools-1.14.4-win32-x64\debugAdapters\bin\WindowsDebugLauncher.exe' '--stdin=Microsoft-MIEngine-In-ci-fosgsc.sig' '--stdout=Microsoft-MIEngine-Out-s2fjk8xa.sld' '--stderr=Microsoft-MIEngine-Error-g1kpw3c.2hm' '--pid=Microsoft-MIEngine-Pid-e4xswjgg.tuq' '--dbgExe=C:\msys64\mingw64\bin\gdb.exe' '--interpreter=mi'
Chcesz zacząć grę?(Tak/Nie)
tak
To zaczynamy!
```

```
Chcesz stworzyć swojej postaci?(Tak/Nie)
tak

Wybierz gatunek/imię pierwszego Ninjago:
Wpisz gatunek: wojownik
Wpisz imię: A

Wybierz moce(obrona/atak/predkosc):
Obrona: 55
Atak: 45
Predkosc: 65
```

```
Chcesz wyświetlić dane pierwszej postaci?(Tak/Nie)
tak

Jestem wojownik, nazywam się A mam takie moce:
obrona = <55>,
atak = <45>,
predkosc = <65>.
```

```
Wybierz gatunek/imie drugiego Ninjago:  
Wpisz gatunek: czarownik  
Wpisz imie: B  
  
Wybierz moce(obrona/atak/predkosc):  
Obrona: 45  
Atak: 75  
Predkosc: 25
```

```
Chcesz wyswietlic dane drugiej postaci?(Tak/Nie)  
tak  
  
Jestem czarownik, nazywam sie B mam takie moce:  
obrona = <45>,  
atak = <75>,  
predkosc = <25>.
```

```
Zaczyna sie pojedynek!  
Dane Ninjago 1:  
Gatunek: wojownik.  
Imie: A.  
Obrona: 55.  
Ataka: 45.  
Predkosc: 65.  
  
Dane Ninjago 2:  
Gatunek: czarownik.  
Imie: B.  
Obrona: 45.  
Ataka: 75.  
Predkosc: 25.  
  
W bitwie!  
  
Wygrywa A!
```

Wygrywa A, temu że atak Ninjago 1 = obrona Ninjago 2, więc decyduje prędkość:
prędkość Ninjago 1 > prędkość Ninjago 2.

2 wariacja(prosta):

```
Chcesz zaczac gry?(Tak/Nie)
tak

To zaczynamy!
```

```
Chcesz stworzyc swojej postaci?(Tak/Nie)
nie

Wybierz pierwszego Ninjago po jego opisu(1, 2, 3):
1. Jestem wojownik, nazywam sie Zugu mam takie moce:
obrona = <58>,
atak = <76>,
predkosc = <34>.

2. Jestem wojownik, nazywam sie Kai mam takie moce:
obrona = <58>,
atak = <71>,
predkosc = <47>.

3. Jestem czarownik, nazywam sie Yang mam takie moce:
obrona = <65>,
atak = <100>,
predkosc = <65>.

Wybieram 1
Wybrales Zugu.
```

```
Wybierz drugiego Ninjago(1, 2):
1. Jestem wojownik, nazywam sie Kai mam takie moce:
obrona = <58>,
atak = <71>,
predkosc = <47>.

2. Jestem czarownik, nazywam sie Yang mam takie moce:
obrona = <65>,
atak = <100>,
predkosc = <65>.

Wybieram 2
Wybrales Yang.
```

```
Zaczyna sie pojedynek!  
Dane Ninjago 1:  
Gatunek: wojownik.  
Imie: Zugu.  
Obrona: 58.  
Ataka: 76.  
Predkosc: 34.  
  
Dane Ninjago 2:  
Gatunek: czarownik.  
Imie: Yang.  
Obrona: 65.  
Ataka: 100.  
Predkosc: 65.  
  
W bitwie!  
  
Wygrywa Yang!
```

source: ...\\Modul 1 - Wprowadzenie do programowania obiektowego\\Labaratorium 1\\Zadanie 4\\zadanie_4.cpp

Zadanie 5

Jak do tej pory mieliśmy wiele obiektów ale tylko jedną klasę. W większych grach zapewne będą pojawiały się różne rodzaje postaci. Opracować zatem trzy klasy, po jednej dla każdego rodzaju postaci. Co można zauważyć przeglądając te klasy? Czy jest zachowana zasada DRY?

Rozwiązania zadania 5

Klasa Human:

```
zadanie_5.cpp
C:\Users\> R > Desktop > Code > CPP > WSB > Semestr 2 > Podstawy Programowania IBrzezek S2 > Modul 1 - Wprowadzenie do programowania obiektowego > Laboratorium 1 > Zadanie 5 > zadanie_5.cpp > ...
1  #include <iostream>
2
3  using namespace std;
4
5  class LegoNinjaboHuman {
6
7  private:
8      string category = "czlowiek";
9      string name;
10     int defence;
11     int attack;
12     int speed;
13
14 public:
15     void wczytajDane(string _category, string _name, int _defence, int _attack, int _speed) {
16
17         category = _category;
18         name = _name;
19         defence = _defence;
20         attack = _attack;
21         speed = _speed;
22     }
23
24     void czytajDane(string &_category, string &_name, int &_defence, int &_attack, int &_speed) {
25
26         _category = category;
27         _name = name;
28         _defence = defence;
29         _attack = attack;
30         _speed = speed;
31     }
32
33     void pokazDane() {
34
35         cout << "Jestem " << category << ", " << "nazywam sie " << name << " mam takie moce: " << endl;
36         cout << "obrona = <" << defence << ">," << endl;
37         cout << "atak = <" << attack << ">," << endl;
38         cout << "predkosc = <" << speed << ">." << endl;
39     }
40
41 };
42
43
44
45
```

Klasa Snake:

```
zadanie_5.cpp x
WSB > Semestr 2 > Podstawy Programowania IBrzezek S2 > Modul 1 - Wprowadzenie do programowania obiektowego > Labaratorium 1 > Zadanie 5 > zadanie_5.cpp > main()
1  #include <iostream>
2
3  using namespace std;
4
5  > class LegoNinjaboHuman { ...
45
46  class LegoNinjaboSnake {
47
48  private:
49      string category = "waz";
50      string name;
51      int defence;
52      int attack;
53      int speed;
54
55  public:
56      void wczytajDane(string _category, string _name, int _defence, int _attack, int _speed) {
57
58          category = _category;
59          name = _name;
60          defence = _defence;
61          attack = _attack;
62          speed = _speed;
63
64      }
65
66      void czytajDane(string &category, string &name, int &defence, int &attack, int &speed) {
67
68          _category = category;
69          _name = name;
70          _defence = defence;
71          _attack = attack;
72          _speed = speed;
73
74      }
75
76      void pokazDane() {
77
78          cout << "Jestem " << category << ", " << "nazywam sie " << name << " mam takie moce: " << endl;
79          cout << "obrona = <" << defence << ">," << endl;
80          cout << "atak = <" << attack << ">," << endl;
81          cout << "predkosc = <" << speed << ">." << endl;
82
83      }
84
85  };
86
87 > class LegoNinjaboProtagonist { ...
127
128 > int main() { ...
```

Klasa Protagonist:

```
zadanie_5.cpp
WSB > Semestr 2 > Podstawy Programowania I Brzezek S2 > Modul 1 - Wprowadzenie do programowania obiektowego > Labaratorium 1 > Zadanie 5 > zadanie_5.cpp > main()
1  #include <iostream>
2
3  using namespace std;
4
5  > class LegoNinjagoHuman { ...
45
46 > class LegoNinjagoSnake { ...
86
87 class LegoNinjagoProtagonist {
88
89 private:
90     string category = "zlodziej";
91     string name;
92     int defence;
93     int attack;
94     int speed;
95
96 public:
97     void wczytajDane(string _category, string _name, int _defence, int _attack, int _speed) {
98
99         category = _category;
100        name = _name;
101        defence = _defence;
102        attack = _attack;
103        speed = _speed;
104    }
105
106
107     void czytajDane(string &category, string &name, int &defence, int &attack, int &speed) {
108
109         _category = category;
110         _name = name;
111         _defence = defence;
112         _attack = attack;
113         _speed = speed;
114     }
115
116
117     void pokazDane() {
118
119         cout << "Jestem " << category << ", " << "nazywam sie " << name << " mam takie moce: " << endl;
120         cout << "obrona = <" << defence << ">," << endl;
121         cout << "atak = <" << attack << ">," << endl;
122         cout << "predkosc = <" << speed << ">." << endl;
123     }
124
125 };
126
127
128 > int main() { ...
```

Co można zauważyć przeglądając te klasy? Czy jest zachowana zasada DRY?

Przeglądając te klasy można zauważyć, że mają one podobną strukturę oraz funkcjonalność. Wszystkie trzy klasy mają prywatne pola przechowujące kategorię, imię, atak, obronę i prędkość.

Każda klasa ma także trzy metody: **wczytajDane**, **czytajDane** i **pokazDane**, które realizują odpowiednio wprowadzanie danych do obiektu, pobieranie danych z obiektu i wyświetlanie informacji o obiekcie.

Nie jest zachowana zasada DRY, ponieważ ta sama funkcjonalność występuje w każdej z klas. Powtarzające się elementy kodu, takie jak pola klasy i metody, powinny być zdefiniowane tylko raz, a następnie odwoływać się do nich w innych klasach.

source: ...\\Modul 1 - Wprowadzenie do programowania obiektowego\\Labaratorium 1\\Zadanie 5\\zadanie_5.cpp