

Języki Programowania, Laboratorium 10

1. Zaimplementować klasę **Figura**, będącą klasą bazową dla rzeczywistych figur rysowanych w programie. Klasa **Figura**, przechowuje następujące pola:

- kolor krawędzi figury - użyć typu string,
- grubość krawędzi figury (grubości linii przyjmują wartości rzeczywiste, typ float),

Pola dostępne są tylko dla klas potomnych.

W części publicznej klasa ta zawiera deklarację wirtualnej metody `Wypisz()` oraz wirtualnej metody `Pole()` (domyślnie powinna zwracać 0).

2. Jako przykład klasy odpowiadającej danemu rodzajowi figur **zaimplementować klasę Okrąg**, będącą klasą potomną klasy **Figura**, zawierającą pola danych potrzebne do odrysowania figury (współrzędne środka, promień) oraz odpowiednie metody. Zdefiniować konstruktor domyślny (wszystkie wartości = 0), jak również konstruktor inicjalizujący wszystkie pola klasy **Okrąg** (wraz z kolorem krawędzi oraz jej grubością).

- Metoda `Wypisz` powinna wypisać na ekran informację "Okrąg kolor grubosc (x,y,R)", gdzie kolejno mamy wartości „x y R” w każdej linijce dla osobnego okręgu.
- Metoda `Pole` powinna zwrócić wartość pola danego okręgu.

3. Napisać program, testujący działanie klasy:

- Stworzyć wskaźnik (wraz z obiektem) `f1` na klasę **Figura** (używając konstruktora domyślnego). Wypisać (używając metody `Wypisz`) oraz wyświetlić na ekranie jej pole.
- Stworzyć obiekt `o1` klasy **Okrąg** (używając konstruktora domyślnego). Wypisać (używając metody `Wypisz`) oraz wyświetlić na ekranie jego pole.
- Stworzyć wskaźnik (wraz z obiektem) `o2` na klasę **Okrąg** (`x = 2, y = 2, R = 5, kolor zielony, grubość 1`). Wypisać oraz wyświetlić na ekranie jego pole.

Następnie należy stworzyć nowy wskaźnik na klasę Figura.

```
Figura* wskfig;
```

I przypisać do niego okręgi `o1` i `o2` oraz wywołać na takim wskaźniku metodę `Wypisz`, np.:

```
wskfig = &o1;
```

```
wskfig->Wypisz();
```

Program należy ponownie skompilować i odpalić. Co się dzieje, jeśli usuniemy słowo `virtual` z klasy **Figura** (należy przetestować i opowiedzieć prowadzącemu)?

Jakie daje nam to możliwości?

- Możemy tworzyć metody przyjmujące referencje, których zachowanie będzie zależne od typu klasy pochodnej.
- Jeśli pojawi się kiedykolwiek konieczność rozszerzenia programu o nowe obiekty, to dodanie nowej klasy będącej pochodną od już istniejącej nie wymaga tysiąca zmian w różnych miejscach programu.

Podsumujmy:

Jeśli kompilator natrafi na **obiekt** danej klasy, np. **Figura**, uruchamia dla niego funkcję składową z właściwej mu klasy.

Jeśli kompilator natrafi na **wskaźnik lub referencję** klasy **Figura**, ma dwie możliwości wywołania funkcji składowej:

1) Skoro jest to wskaźnik do klasy **Figura**, wywołuje metodę składową klasy **Figura**. Jest to domyślne zachowanie kompilatora.

2) Kompilator widzi, że jest to wskaźnik do klasy **Figura**, ale zamiast na ślepo sięgać do klasy **Figura** *używa swojej inteligencji*: nie daje się zwieść typem i sprawdza, na co faktycznie wskaźnik wskazuje. Wtedy może się zorientować, że wskaźnik tak naprawdę wskazuje na obiekt klasy pochodnej **Okrąg**, zatem **orientując się według typu obiektu** uruchamia funkcję składową okręgu. Takie zachowanie wymusza słowo kluczowe virtual przed nazwą funkcji składowej.

4. Następnie **zamieniamy metodę wirtualną z klasy podstawowej na czysto wirtualną** (dość bezsensowne jest liczenie pola dla „figury”).

```
virtual double Pole() = 0;
```

Należy odpowiednio poprawić kod.

Przypomnienie:

Jeśli klasa deklaruje jedną ze swoich funkcji jako virtual, wówczas jej destruktor deklarujemy także jako virtual.

5. Należy dopisać do klasy **Okrąg** metody `Zapisz` oraz `Wczytaj`:

- Metoda `Zapisz(char*)` powinna zapisać do pliku, którego nazwa podana jest jako jej parametr, współrzędne środka okręgu oraz promień.

Zapisywanie do pliku:

```
#include <fstream>

ofstream ofile;
ofile.open("file.txt");
ofile<<"aaa"<<123<<endl;
ofile.close();
```

- Zapisać dane obiektu o2 do pliku „figury.txt”.
- Metodę `Wczytaj(char* a, int b)` powinna wczytać z pliku, którego nazwa podana jest jako jej parametr (a), z linii o numerze podanym jako drugi parametr (b), współrzędne środka okręgu oraz promień, oraz przypisać je odpowiednim składnikom `Okręgu`. Przykład pliku: `figury.txt`:

```
4 5 6
2 3 8
3 2 7
```

Wczytywanie z pliku (np. liczby):

```
#include <fstream>

fstream ifile; int val;
ifile.open("figury2.txt");
while(ifile>>val)
{
    cout<<"val: "<<val<<endl;
}
ifile.close();
```

- Wczytać do obiektu `o1` dane z drugiej linii z przykładowego pliku „figury2.txt”. Wypisać. Przykładowy plik należy stworzyć samodzielnie.
6. Pamiętać o napisaniu `makefile'a`. `Makefile`, powinien umożliwiać kompilację napisanego programu, definiować zmienną `CXX` określającą kompilator (g++) oraz `CXXFLAGS` definiującą flagę (-Wall), a tworzone klasy powinny być wstępnie kompilowane do plików *.o.
 7. Zamiast do pliku `figury.txt` należy zapisać figurę do pliku, którego nazwa jest podana jako **pierwszy parametr wywołania programu**. Należy mu również ustawić promień okręgu na podany jako **drugi parametr wywołania programu** (uwaga, program powinien wypisywać odpowiednie ostrzeżenie, jeśli liczba podanych parametrów jest nieprawidłowa!)

Parametry wywołania programu (zmiany w deklaracji funkcji `main`): `int main(int argc, char **argv){ ... }`, gdzie:

`argc` - liczba parametrów, z którymi został wywołany program

`**argv` - tablica parametrów (każdy jako `char*`):

`argv[0]` - nazwa programu

`argv[1], argv[2]...` - kolejne parametry wywołania programu

Np. wywołanie programu mogło by mieć formę:

```
./program Ala 3
```

Wtedy:

```
argc == 3
```

```
argv[0] == "program"
```

```
argv[1] == "Ala"
```

```
argv[2] == "3" // zmiana "3" na 3 (liczbę) – funkcja atoi. np int a = atoi("3"); z biblioteki <stdlib.h>
```

Przyporządkowanie tych parametrów wykonuje się samoistnie w momencie wywołania programu, jedynym wkładem programisty jest zadeklarowanie funkcji `main` w formie `int main(int argc, char **argv)`