Programowanie obiektowe

Zajęcia 2. Konstruktury, dziedziczenie, przekazywanie przez wartość/referencję

Zadanie 1. Projekt konstruktor

1. Utwórz w wybranym miejscu na dysku folder `ConstructorCpp`.
2. W VSC otwórz utworzony w poprzednim punkcie folder.
3. Do projektu dodaj plik `main.cpp` oraz plik źródłowy (cpp) oraz nagłówkowy (h) o nazwie wynikającej z klasy implementowanej na poprzednich zajęciach (`Ul.cpp` i `Ul.h`, `ElektrowniaWeglowa.cpp` oraz `ElektrowniaWeglowa.h`, `RadaNadzorcza.cpp` i `RadaNadzorcza.h` albo `JednorekiBandyta.cpp` i `JednorekiBandyta.h`).
4. W pliku `main.cpp` dodaj przykładową pustą implementację funkcji main.
5. Skompiluj i uruchom plik `main.cpp` w trybie debug np. poprzez wciśnięcie przycisku F5 a następnie wybierz odpowiednią dla Twojego systemu i kompilatora konfigurację.
6. Zmodyfikuj użytą konfigurację (zapisaną w pliku `.vscode/tasks.json`) w taki sposób żeby zamiast pojedynczego pliku kompilacji uległy wszystkie pliki cpp zawarte w katalogu, w którym znajduje się aktualnie przez nas modyfikowany plik. Możesz tego dokonać poprzez zamianę

użytego w pliku `${file}` na `${fileDirname}/*.cpp` lub nieco ogólniej `${fileDirname}/*${fileExtname}`. Dodatkowo jeśli zależy Ci na minimalizacji liczby powstałych w wyniku kompilacji plików wykonywalnych warto w pliku `.vscode/tasks.json` i `.vscode/launch.json` ustalić pojedynczą nazwę pliku wykonywalnego dla tego projektu.

7. Wszystkie pola implementowanej klasy zmodyfikuj w taki sposób, żeby możliwy był dla nich lazy loading, tj. niech stanowią wskaźniki na miejsce w pamięci przechowujące wartość przechowywanej w polu właściwości.

8. Utwórz konstruktor domyślny dla implementowanej klasy inicjalizujący wszystkie wskaźniki (przy pomocy listy inicjalizacyjnej) na wartość `nullptr`.

9. Utwórz destruktor dla implementowanej klasy usuwający obiekty przechowywane przez wskaźniki pól (pod warunkiem, że zostały ustawione).

10. Utwórz konstruktor kopiujący dla implementowanej klasy dokonujący głębokiej kopii wartości wskazywanych przez pola.

11. Utwórz konstruktor przenoszący dla implementowanej klasy zgodnie z założeniami semantyki move.

12. Utwórz konstruktor konwertujący obiekty typu jednego z pól klasy na obiekt implementowanej klasy (np. literał ciągu znaków będącego nazwą pasieki na obiekt klasy `Ul`).

13. Zaimplementuj funkcjonalność klasy zgodnie z opisem z zeszłych zajęć.

14. Utwórz funkcję przyjmującą **przez wartość** obiekt implementowanej klasy. Np.:

```
void foo(Ul ul) {
  //... (implementacja)
}
```

15. Utwórz obiekt implementowanej klasy, wywołaj metody ustawiające wartości poszczególnych pól, a następnie przekaż obiekt do utworzonej w poprzednim punkcie funkcji. Upewnij się poprzez skorzystanie z trybu debug, że został wywołany konstruktor kopiujący.

16. Jeszcze raz wywołaj funkcję z punktu 14, tym razem użyj semantyki `move` celem przeniesienia wcześniej utworzonego obiektu do parametru funkcji. Upewnij się poprzez skorzystanie z trybu debug, że został wywołany odpowiedni konstruktor.

17. Ponownie wywołaj funkcję z punktu 14, tym razem zamiast obiektu implementowanej klasy przekaż do funkcji obiekt zdolny do bycia skonwertowanym na obiekt implementowanej klasy. Upewnij się poprzez skorzystanie z trybu debug, że został wywołany odpowiedni konstruktor.

18. Utwórz jeszcze jedną funkcję, tym razem przyjmującą obiekt implementowanej klasy przez "stałą" referencję L-value, np.:

```
void bar (const Ul &ul) {
  //... (implementacja)
}
```

19. Spróbuj wykonać kroki od 15 do 17 tym razem przekazując obiekt w różnych wariantach do funkcji utworzonej w punkcie 18. Zapisz swoje obserwacje i spróbuj je zinterpretować.

20. Utwórz kolejną funkcję przyjmującą obiekt implementowanej klasy - tym razem w wersji "zwykłej" referencji L-value, np.:

```
void baz (Ul &ul) {
  //... (implementacja)
}
```

21. Powtórz próby wywołania funkcji z przekazaniem obiektu w trzech wariantach (15-17), zapisz obserwacje i spróbuj je zinterpretować.

22. Utwórz ostatnią już wersję funkcji przyjmującą obiekt implementowanej klasy - tym razem w wersji referencji R-value, np.:

```
void qux (Ul &&ul) {
  //... (implementacja)
}
```

23. Powtórz próby wywołania funkcji z przekazaniem obiektu w trzech wariantach (15-17), zapisz obserwacje i spróbuj je zinterpretować.

24. Utwórz w projekcie klasę `Logger` (utwórz dla niej plik źródłowy i nagłówkowy). Zaimplementuj w niej publiczną metodę `log` przyjmującą ciąg znaków. Metoda powinna poprzedzić wypisywany na standardowe wyjście ciąg znaków aktualną datą (jak najdokładniejszą jak się da).

25. Wydziedzicz implementowaną wcześniej klasę z klasy `Logger` w następujący sposób:

```
class Ul: Logger {
  //... (implementacja)
};
```

26. Czy z poziomu funkcji main możesz wywołać metodę `log` obiektu implementowanej klasy? Napisz szerszy komentarz w tej kwestii.

27. Wewnątrz implementowanej klasy skorzystaj z metody log celem określenia, które spośród jej metod zostały wywołane.

28. W klasie `Logger` dodaj konstruktor przyjmujący parametr `debug` typu bool, który określa czy metoda log powinna wypisywać wiadomości logu na standardowe wyjście czy też nie.

29. Czy teraz możesz skompilować projekt? Postaraj się ustalić przyczynę problemu, opisz ją a następnie postaraj się zmodyfikować kod tak żeby projekt znowu się kompilował.

30. Utwórz w projekcie klasę `Stopper` (utwórz dla niej plik źródłowy i nagłówkowy). Zaimplementuj w niej funkcjonalność startowania, stopowania, resetowania, oraz wypisywania sumarycznego odmierzonego czasu.

31. Wydziedzicz implementowaną wcześniejszą klasę dodatkowo z klasy `Stopper`. Podczas uruchamiania konstruktora zresetuj stoper. Podczas uruchamiania którejkolwiek z metod implementowanej klasy startuj stoper a tuż przed zakończeniem jej działania dokonaj jego stopowania. Przetestuj działanie stopera uruchamiając metody implementowanej klasy w pętli i wywołanie wypisania sumerycznego odmierzonego czasu stopera.

32. Zauważ, że w poprzednim punkcie sprawiłeś że implementowana wcześniej klasa dziedziczy z dwóch klas. Zastanów się jakie ryzyka to za sobą niesie. Zweryfikuj czy w języku Java również jest możliwość wielokrotnego dziedziczenia, a jeśli nie - co ją zastępuję.