Imię i nazwisko: | | | | | | | | | | | | | | | | | | | | | | | | | | **25L kol 1**

Proszę założyć, że udostępniona jest przestrzeń nazw std

Zadanie 1 - obiektowe wzorce projektowe (3pkt)

Obiekty klas pochodnych po `Member` reprezentują członków komisji `Committee`. W komisji mogą być studenci, są oni poprawnie reprezentowani przez obiekty typu `Student`, klasa ta nie jest w hierarchii `Member` i klasy tej nie można zmienić.

```
class Member {
public:
    virtual ~Member() {}
    virtual void accept(Visitor& v) const = 0;
};
using VMember = Member*; //widok na obiekt, nie zarzadza obiektem
//Powinno byc 'using VMember = reference_wrapper<Member>', ale w tym zadaniu poprzednia definicja wystarczy

using Committee = vector<VMember>;
class Admin : public Member {
public:
    void accept(Visitor& v) const override { v.visit(*this); }
};
class Expert : public Member {
public:
    void accept(Visitor& v) const override { v.visit(*this); }
};

//reprezentuje studenta, ma odpowiednia funkcjonalnosc, nie mozna zmieniac tej klasy
class Student {
};
```

Uzupełnij kod, dodając klasy lub funkcje poniżej i dostarcz funkcję `checkBalanced`. Funkcja ta bada, czy w komisji jest przynajmniej jeden student, przynajmniej jeden przedstawiciel administracji i przynajmniej jeden ekspert.

```
bool checkBalanced(const Committee& c) {
```

Zadanie 2 - obiektowe wzorce projektowe (1pkt)

Prostokąt przedstawiony jest zależny w części implementacyjnej od bibliotek graficznych, jest to potrzebne do implementacji funkcji 'draw'. Udało się przenieść implementację rysowania do pliku CPP, niestety zależności zostały (np. składowa Window*). Popraw implementację ukrywając zależności w pliku CPP.

PLIK HPP

```
// Naglowek Window.hpp jest poprzebny tylko do implementacji.
// Nie jest czescia interfejsu klas Figure, Rect.
#include "Window.hpp"

class Figure {
public:
    virtual void draw() const = 0;
};
class Rect : public Figure {
public:
    void draw() const override {
        doDraw();
    }
private:
    void doDraw() const;

    //uchwyt do okna, do piora i inne skladowe niezebedne do rysowania
    Window* w;
};
```

PLIK CPP

```
void Rect::doDraw() const {
       //bardzo skomplikowane i uzywa wielu bibliotek zewnetrznych
}
```

Uwagi do prowadzącego (R. Nowaka):

Zadanie 3 - cykl życia oprogramowania (2pkt)

Proszę odpowiedzieć na następujące pytania:

1. W którym momencie procesu wytwarzania oprogramowania uzyskujemy gwarancję poprawności programu? Kiedy ten proces się kończy?

2. Jak można wykorzystać etap zbierania wymagań do uniknięcia konfliktów? Jak połączyć to z innymi elementami cyklu życia oprogramowania?

Zadanie 4 - SOLID (2pkt)

Czy fragment kodu pokazany niżej jest zgodny z regułami SOLID? Jeśli nie, to w jaki sposób je narusza (może naruszać więcej niż jedną regułę)? Jak należałoby poprawić kod?

```
class Page
{
public:
  const std::vector<std::string>& words() const;

  void mergeWith(const Content& other)
  {
    words_.append_range(other.words());
  }

  bool isSpellingCorrect() const
  {
    EnglishDictionary dictionary;
    return std::ranges::all_of(words(),
                               [&dictionary](const auto& w) {
                                 return dictionary.hasElement(w);
                               });
  }

  virtual void print(Printer& printer)
  {
     // ...
  }

protected:
  std::vector<std::string> words_;
};

class TitlePage : public Page
{
public:
  explicit HiddenContent()
  {
    words_.emplace_back("TITLE");
  }

  void print(Printer& printer) override
  {
    throw std::runtime_error("Custom␣printer␣required");
  }
};
```

Zadanie 5 - Wymagania (2pkt)

Proszę zaproponować kilka wymagań dla systemu wspomagającego zapisy studentów na laboratoria. Zbiór nie musi być kompletny, ale musi być spójny.

Uwagi do prowadzącego (K. Grochowskiego):