Imię i nazwisko:..... **23L kol 1**

Proszę założyć, że udostępniona jest przestrzeń nazw std

Zadanie 1 - obiektowe wzorce projektowe (2pkt)

Wywołanie metody `close` dla okna nadrzędnego (`MainView`) powinno uruchamiać metody `close` dla wszystkich otwartych okien podrzędnych (obiekty `ChildViewA` lub `ChildViewB`). Zakładamy, że okno jest otwarte, gdy wywołano metodę `open`. Uzupełnij implementację klas `MainView`, `ChildViewA` i `ChildViewB`. Obok test.

```
int main() {
  MainView m;
  m.open();
  ChildViewA ch1(&m);
  ch1.open();
  ChildViewB ch2(&m);
  ch2.open();
  ChildViewA ch3(&m);
  m.close();
}
```

```
class MainView {
public:
    void open(){}
    void close(){}
};


class ChildViewA {
public:
    ChildViewA(MainView* m) : m_(m) {}
    void open(){}
    void close(){}
private:
    MainView* m_;
};

class ChildViewB {
public:
    ChildViewB(MainView* m) : m_(m) {}
    void open(){}
    void close(){}

private:
    MainView* m_;
};
```

Uwagi do prowadzącego (R. Nowaka):

Zadanie 2 - obiektowe wzorce projektowe (2pkt)

Popraw funkcję `send_mssages`. Wysyła ona komunikaty. Wykorzystaj abstrakcyjny komunikat oraz abstrakcyjną klasę do wysyłania i do odczytywania potwierdzeń. Nie zmieniaj klas `Message`, `Sender`, `Confirm` oraz ich klas pochodnych.

```
class Message { //abstrakcyjany komunikat
public:
    Message(const std::string& msg) : msg_(msg){}
    virtual ~Message() {}
    const string& get() const { return msg_; }
private:
    string msg_;
};
using PMessage = shared_ptr<Message>;
class MessageSMTP : public Message {
public:
    MessageSMTP(const string& msg) : Message(msg){}
};
class MessageTCP : public Message {
public:
    MessageTCP(const string& msg) : Message(msg){}
};
```

```
class Sender { //abstrakcyjana klasa nadajnika komunikatow
public:
    virtual void connect() = 0;
    virtual void send(PMessage m) = 0;
    virtual ~Sender() {}
};
using PSender = shared_ptr<Sender>;
class SenderSMTP : public Sender {
public:
    virtual void connect();
    virtual void send(PMessage m);
};
class SenderTCP : public Sender {
public:
    virtual void connect();
    virtual void send(PMessage m);
};
class Confirm { //abstrakcyjana klasa do potwierdzen
public:
    virtual void connect() = 0;
    virtual int getNumberSent() = 0;
    virtual ~Confirm() {}
};
using PConfirm = shared_ptr<Confirm>;
class ConfirmSMTP : public Confirm {
public:
    virtual void connect();
    virtual int getNumberSent();
};
class ConfirmTCP : public Confirm {
public:
    virtual void connect();
    virtual int getNumberSent();
};
```

```
enum PROTOCOL { SMTP, TCP };
bool send_messages(PROTOCOL protocol, const vector<string>& msgs) {
    PSender sender;
    switch(protocol) {
    case SMTP:
        sender = PSender(new SenderSMTP);
        break;
    case TCP:
        sender = PSender(new SenderTCP);
        break;
    }
    for(std::string s : msgs) {
        PMessage m;
        switch(protocol) {
        case SMTP:
            m = PMessage(new MessageSMTP(s));
            break;
        case TCP:
            m = PMessage(new MessageTCP(s));
            break;
        }
        sender->send(m);
    }
    PConfirm confirm;
    switch(protocol) {
    case SMTP:
        confirm = PConfirm(new ConfirmSMTP);
        break;
    case TCP:
        confirm = PConfirm(new ConfirmTCP);
        break;
    }
    confirm->connect();
    if( confirm->getNumberSent() == msgs.size() )
        return true;
    else
        return false;
}
```

Zadanie 3 - cykl życia oprogramowania (2pkt)

Proszę odpowiedzieć na następujące pytania:

1. Kiedy można powiedzieć o wymaganiu, że jest jednoznaczne? Podaj przykład - własny, wskazujący na zrozumienie tematu.

2. Dlaczego przyjmuje się, że dobry test powinien być deterministyczny? Podaj przykład problemu i jego rozwiązania.

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?

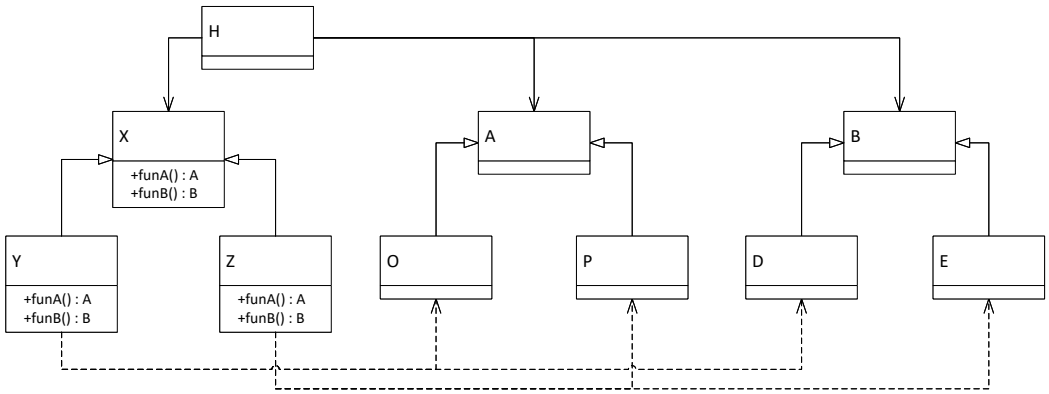
```
class Button
{
public:
  Button(chrome::Session& session_, std::string_view url_)
    : session(session_), url(url_)
  {}
  virtual ~Button() {}
  virtual void render(Screen& screen)
  {
    // ...
  }

  void onClick()
  {
    session.request(url + "/pressed");
  }
private:
  chrome::Session& session;
  std::string url;
};
class HiddenButton : public Button
{
public:
  HiddenButton(chrome::Session& session, std::string_view url)
    : Button(session, url) {}

  void render(Screen& screen) override
  {
    throw std::runtime_error("Button␣hidden");
  }
};
```

Zadanie 5 - UML (2pkt)

Opisz wszystkie relacje między klasami widocznymi na diagramie. Jeśli diagram przypomina znajomy wzorzec, zaproponuj ciekawsze nazwy klas (inne niż na wykładzie).



Uwagi do prowadzącego (K. Grochowskiego):