Imię i nazwisko: | | | | | | | | | | | | | | | | | | | | | | | | | | **23Z kol 1**

Proszę założyć, że udostępniona jest przestrzeń nazw std

Zadanie 1 - obiektowe wzorce projektowe (2pkt)

Hierarchia wyjątków BaseException w aplikacji dostarcza wzorca Wizytatora. Obsługa błędu powinna zliczać obiekty, osobno dla każdej z klas konkretnych. W tym celu stworzono klasę Counters. Popraw implementację obsługi błędów, aby ją znacznie przyspieszyć.

```
class Visitor {
public:
    virtual ~Visitor() {}
    virtual void visit(NetworkError&) = 0;
    virtual void visit(UserBreak&) = 0;
    virtual void visit(BadData&) = 0;
};
```

```
class BaseException : public std::exception {
public:
    virtual void accept(Visitor& v) = 0;
};
class NetworkError : public BaseException {
public:
    virtual void accept(Visitor& v) { v.visit(*this); }
};
class UserBreak : public BaseException {
public:
    virtual void accept(Visitor& v) { v.visit(*this); }
};
class BadData : public BaseException {
public:
    virtual void accept(Visitor& v) { v.visit(*this); }
};
```

```
class Counters {
public:
    static Counters& getInstance() { //ta metoda jest poprawna!
        static Counters instance; //obiekt lokalny statyczny, tak mozna dostarczyc singleton
        return instance;
    }
    int getNetworkError() const { return networkError_; }
    int getUserBreak() const { return userBreak_; }
    int getBadData() const { return badData_; }
    void incNetworkError() { ++networkError_; }
    void incUserBreak() { ++userBreak_; }
    void incBadData() {++badData_; }
private:
    Counters() : networkError_(0), userBreak_(0), badData_(0) {}
    int networkError_;
    int userBreak_;
    int badData_;
};

int main() {
    try {
        //kod ktory moze zglosic wyjatek typu BaseException
    }
    catch(NetworkError& e) {
        Counters::getInstance().incNetworkError();
    }
    catch(UserBreak& e) {
        Counters::getInstance().incUserBreak();
    }
    catch(BadData& e) {
        Counters::getInstance().incBadData();
    }
    return 0;
}
```

Uwagi do prowadzącego (R. Nowaka):

Zadanie 2 - obiektowe wzorce projektowe (2pkt)

Dostarcz fabrykę, która tworzy obiekty Leopard("1"), Leopard("2"), Leopard("2E"), Abrams("M1"), Abrams("M1A1") na podstawie identyfikatora Tank::Id i dostarcza uchwyt do klasy bazowej (obiekt typu UTank). Przykład użycia:

```
UTank tank1 = TankFactory::getInstance().create(Tank::Id::LEOPARD_1);
cout << tank1->getVersion() << endl;
```

```
using UTank = unique_ptr<Tank>;
class Tank {
public:
    enum Id { LEOPARD_1, LEOPARD_2, LEOPARD_2E, ABRAMS_M1, ABRAMS_M1A1 };
    Tank(const string& version) : version_(version) {}
    virtual ~Tank() {}
    const string& getVersion() const { return version_; }
private:
    string version_;
};

class Leopard : public Tank {
public:
    Leopard(const string& version) : Tank(version) {}
};

class Abrams : public Tank {
public:
    Abrams(const string& version) : Tank(version) {}
};

class TankFactory {
public:
    class UnknownIdError : public std::exception {};

    static TankFactory& getInstance() { //ta metoda powinna zostac, jest poprawna
        static TankFactory instance;
        return instance;
    }

    UTank create(Tank::Id id) const {
        throw UnknownIdError();
    }
private:
    TankFactory() {}
};
```

Zadanie 3 - cykl życia oprogramowania (2pkt)

Proszę odpowiedzieć na następujące pytania:

1. W którym momencie procesu wytwarzania oprogramowania uzyskujemy gwarancję poprawności programu? Kiedy możemy go dostarczyć klientowi?

2. Dlaczego warto oczekiwać, by wymaganie było, między innymi, weryfikowalne i jednoznaczne?

Zadanie 4 - SOLID (2pkt)

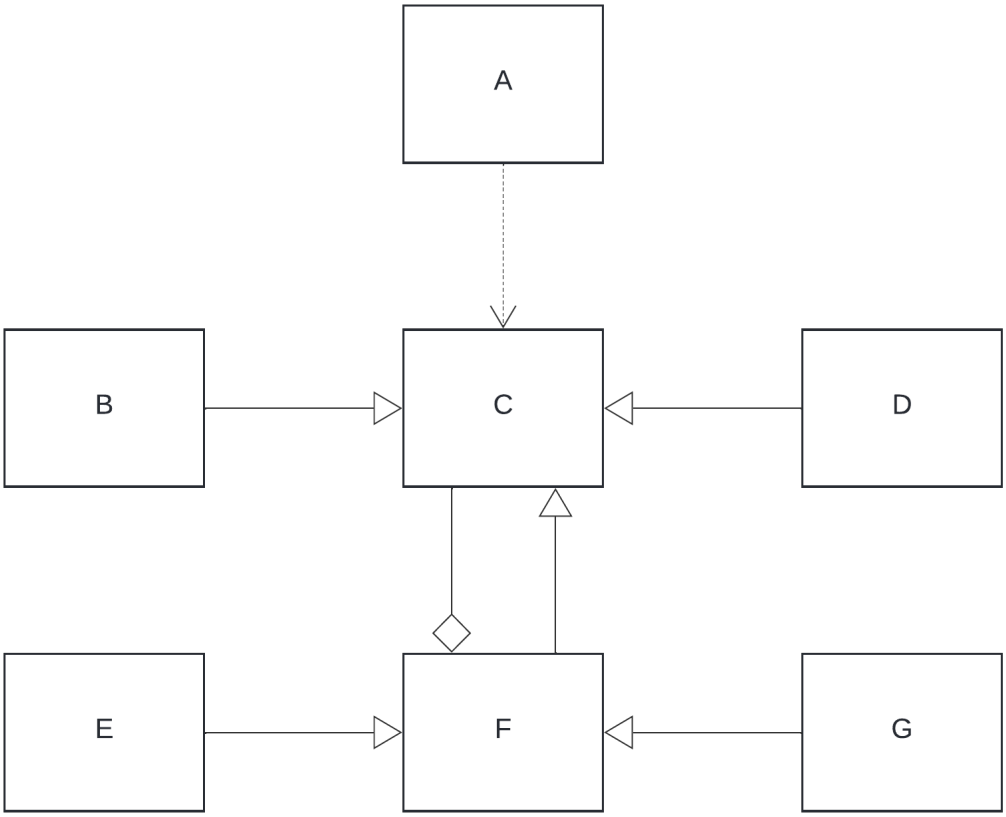
Czy fragment kodu pokazany niżej jest zgodny z regułami SOLID? Jeśli nie, to w jaki sposób je narusza (może naruszać więcej niż jedną regułę)? Jak należałoby poprawić kod?

```
class Content
{
public:
  const std::vector<std::string>& words() const;
  void mergeWith(const Content& other)
  {
    words_.insert(std::end(words_),
                  std::cbegin(other.words()), std::cend(other.words()));
  }
  bool isSpellingCorrect() const
  {
    EnglishDictionary dictionary;
    return std::ranges::all_of(words(),
                               [&dictionary](const auto& w) {
                                 return dictionary.hasElement(w);
                               });
  }
  virtual void render(Screen& screen)
  {
     // ...
  }
protected:
  std::vector<std::string> words_;
};

class HiddenContent : public Content
{
public:
  explicit HiddenContent()
  {
    words_.emplace_back("Hide!");
  }
  void render(Screen& screen) override
  {
    throw std::runtime_error("Content␣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):