Imię i nazwisko: | | | | | | | | | | | | | | | | | | | | | | | | | | **26L kol 1**

Proszę założyć, że udostępniona jest przestrzeń nazw std

Zadanie 1 - obiektowe wzorce projektowe (2 pkt)

Obiekty `File` reprezentuje plik, zaś `Dir` katalog w różnych systemach plików (FAT16, NTFS, EXT4). Obiekty te tworzy fabryka `Factory`. Mamy także klasy `FileFat16`, `DirFat16`, ..., można je modyfikować, podobnie jak `File`, `Dir`.

(1) Proszę zadbaj o to, aby pliki i katalogi tworzone przez fabrykę były z tego samego systemu plików.

(2) Udostępnij możliwość reprezentacji drzewiastych struktur katalogów, katalog może mieć podkatalogi, a nie tylko pliki jak obecnie.

W kodzie użyto wskaźnika typu `std::shared_ptr`. Nie ma to wpływu na zadanie. Traktuj ten obiekt jak wskaźnik.

```
using PFile = shared_ptr<File>;
using PDir = shared_ptr<Dir>;
class Factory {
public:
    virtual PFile createFile(const string& name) = 0;
    virtual PDir createDir(const string& name) = 0;
  };
```

```
class File {
public:
    virtual ~File() = default;
};
class Dir {
public:
    virtual ~Dir() = default;
    vector<PFile> children_;
};
class FileFat16 : public File {
public:
    FileFat16(const string& name){}
};

class DirFat16 : public Dir {
public:
    DirFat16(const string& name){}
};

class FileNTFS : public File {
public:
    FileNTFS(const string& name){}
};

class DirNTFS : public Dir {
public:
    DirNTFS(const string& name){}
};

class FileExt4 : public File {
public:
    FileExt4(const string& name){}
};

class DirExt4 : public Dir {
public:
    DirExt4(const string& name){}
};
```

Zadanie 2 - obiektowe wzorce projektowe (2pkt)

Obiekty klas pochodnych po `Element` reprezentują katalog (Dir) lub plik (File). Uzupełnij kod funkcji `getSize`, która zwraca wielkość kolekcji elementów. Plik przechowuje swoją wielkość (ma metodę `getSize`), zaś katalog ma wielkość będącą sumą wielkości obiektów podrzędnych oraz 1024B (dodatkowy narzut na katalog, nawet gdy jest pusty).

Używamy `std::size_t`, można traktować to tutaj jak `int`.

```
class Element {
public:
    virtual ~Element() = default;
    virtual void accept(Visitor& v) const = 0;
};
using VElement = Element*; //widok na obiekt, nie zarzadza obiektem
//Powinno byc (C++17): using VElement = reference_wrapper<Element>
//ale w tym zadaniu poprzednia definicja wystarczy
using Elements = vector<VElement>;

class Visitor {
public:
    virtual void visit(const File& f) = 0;
    virtual void visit(const Dir& d) = 0;
};
class File : public Element {
public:
    File(std::size_t size) : size_(size) {}
    void accept(Visitor& v) const override { v.visit(*this); }
    std::size_t getSize() const { return size_; }
private:
    std::size_t size_;
};

class Dir : public Element {
public:
    Dir() {}
    void accept(Visitor& v) const override { v.visit(*this); }

    const Elements& getChildren() const { return children_; }
private:
    Elements children_;
};
```

```
std::size_t getSize(const Elements& v) {
```

Uwagi do prowadzącego (R. Nowaka):

Zadanie 3 - wytwarzanie oprogramowania (2pkt)

Proszę odpowiedzieć na następujące pytania:

1. Podaj dwa przykłady: wymagania, które można próbować zaimplementować z wykorzystaniem TDD i takiego, którego nie powinno się tak implementować.

2. Po co definiuje się metody weryfikacji wymagań? O jakie metody warto zabiegać i dlaczego?

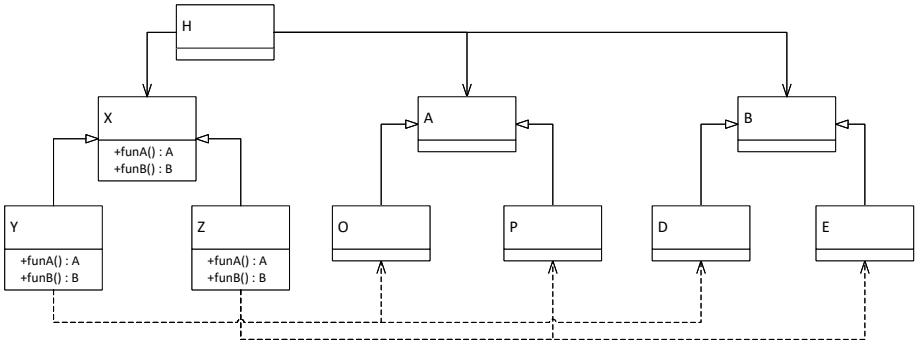
Zadanie 4 - SOLID (2pkt)

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 Series
{
public:
  virtual void load(std::ifstream& in) {
    std::ranges::copy(std::istream_iterator<int>(in), std::istream_iterator<int>(),
    std::back_insert_iterator(items_));
  }

  bool matches(std::unique_ptr<Predicate> predicate) const {
    return std::ranges::all_of(items_, [&predicate](const auto& i) {
        return predicate->check(i);
    });
  }

private:
  std::vector<int> items_;
};

class NamedSeries : public Series
{
public:
  void load(std::ifstream& in) override {
    in >> name_;
    Series::load(in);
    in.close();
  }

private:
  std::string name_;
};
```

Zadanie 5 - UML (2pkt)

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



Uwagi do prowadzącego (K. Grochowskiego):