main.cpp

#include <cstdlib>

#include <iostream>

#include <list>

#include <vector>

#include "collection\_g.h"

#include "exceptions.h"

#include "produit.h"

**using** **namespace** std**;**

**int** main**()** **{**

**{**

cout **<<** "------------------------------------------------------------" **<<** endl**;**

cout **<<** "Test sur Collection<char, vector> :" **<<** endl**;**

**try** **{**

Collection**<char,** vector**>** c**;**

**for** **(char** ch **=** 'A'**;** ch **<** 'D'**;** **++**ch**)**

c**.**ajouter**(**ch**);**

cout **<<** c **<<** " (taille = " **<<** c**.**taille**()** **<<** ")" **<<** endl**;**

c**.**get**(**0**)** **=** 'B'**;**

c**.**get**(**1**)** **=** c**.**get**(**2**);**

c**.**get**(**2**)** **=** 'D'**;**

cout **<<** c **<<** " (taille = " **<<** c**.**taille**()** **<<** ")" **<<** endl**;**

cout **<<** boolalpha

**<<** c**.**contient**(**'A'**)** **<<** endl

**<<** c**.**contient**(**'D'**)** **<<** endl

**<<** noboolalpha**;**

c**.**vider**();**

cout **<<** c **<<** " (taille = " **<<** c**.**taille**()** **<<** ")" **<<** endl**;**

cout **<<** c**.**get**(**0**)** **<<** endl**;**

**}** **catch** **(const** IndiceNonValide**&** e**)** **{**

cout **<<** e**.**what**()** **<<** endl**;**

**}**

cout **<<** "------------------------------------------------------------" **<<** endl**;**

cout **<<** endl**;**

**}**

**{**

cout **<<** "------------------------------------------------------------" **<<** endl**;**

cout **<<** "Test sur Produit :" **<<** endl**;**

**try** **{**

// un produit se caractérise par un no, un libellé, un prix

Produit p1**(**1**,** "Produit 1"**,** 0.05**);**

cout **<<** p1 **<<** endl**;**

**{**

**try** **{**

Produit p2**(**2**,** "Produit 2"**,** 0**);**

**}** **catch** **(const** PrixNonValide**&** e**)** **{**

cout **<<** e**.**what**()** **<<** endl**;**

**}**

**}**

p1**.**setPrix**(**0.0**);**

**}** **catch** **(const** PrixNonValide**&** e**)** **{**

cout **<<** e**.**what**()** **<<** endl**;**

}

cout << "------------------------------------------------------------" << endl;

cout << endl;

}

{

cout << "------------------------------------------------------------" << endl;

cout << "Test sur Collection<Produit, list> :" << endl;

try {

Collection<Produit, list> c;

Produit p1(1, "Produit 1", 1.55);

Produit p2(2, "Produit 2", 5);

c.ajouter(p1);

c.ajouter(p2);

cout << c << " (taille = " << c.taille() << ")" << endl;

Produit tmp = c.get(0);

c.get(0) = c.get(1);

c.get(1) = tmp;

cout << c << " (taille = " << c.taille() << ")" << endl;

cout << boolalpha

<< c.contient(p1) << endl

<< c.contient(p2) << endl

<< noboolalpha;

{

//< à compléter 1 >

auto FnMajoration = [](Produit& p){

p.majoration(10);

};

// On parcourt la collection en majorant le prix de chacun

// des produits de 10%

c.parcourir(FnMajoration);

cout << c << " (taille = " << c.taille() << ")" << endl;

}

c.vider();

cout << c << " (taille = " << c.taille() << ")" << endl;

} catch (const IndiceNonValide& e) {

cout << e.what() << endl;

}

cout << "------------------------------------------------------------" << endl;

cout << endl;

}

return EXIT\_SUCCESS;

}

// ------------------------------------------------------------

// Test sur Collection<char, vector> :

// [A, B, C] (taille = 3)

// [B, C, D] (taille = 3)

// false

// true

// [] (taille = 0)

// Erreur dans Collection::get :

// n doit etre strictement plus petit que collection.size()

// ------------------------------------------------------------

//

// ------------------------------------------------------------

// Test sur Produit :

// (1, "p", 0.05)

// Erreur dans Produit::Produit :

// le prix doit etre >= 5 cts !

// Erreur dans Produit::setPrix :

// le prix doit etre >= 5 cts !

// ------------------------------------------------------------

//

// ------------------------------------------------------------

// Test sur Collection<Produit, list> :

// [(1, "Produit 1", 1.55), (2, "Produit 2", 5.00)] (taille = 2)

// [(2, "Produit 2", 5.00), (1, "Produit 1", 1.55)] (taille = 2)

// true

// true

// [(2, "Produit 2", 5.50), (1, "Produit 1", 1.71)] (taille = 2)

// [] (taille = 0)

// ------------------------------------------------------------

exceptions.h

/\*

-----------------------------------------------------------------------------------

Laboratoire : 04

Fichier : exception.h

Auteur(s) : Paul Reeve, Maxime Scharwath, Thibault Seem

Date : 20.04.2020

But : Implémentation des classes d'exception pour les classes Collections et

Produit qui héritent de std::logic\_error.

La classe IndiceNonValide contient les méthode suivantes :

- IndiceNonValide(const char\* s) : Constructeur pour sa classe.

Ce constructeur ne fait qu'appeler logic\_error de la libraire

stdexept, en lui passant un caratère qu'on souhaite remonter

avec l'exception.

- ndiceNonValde(const std::string& s) : Surcharge du constructeur.

Permet de passer un string à la place d'un caractère simple.

La classe PrixNonValide contient les méthodes suivantes :

- PrixNonValide(const char\* s) : Constructeur pour sa classe.

Ce constructeur ne fait qu'appeler logic\_error de la libraire

stdexept, en lui passant un caratère qu'on souhaite remonter

avec l'exception.

- PrixNonValide(const std::string& s) : Surcharge du constructeur.

Permet de passer un string à la place d'un caractère simple.

Remarque(s) : Les constructeurs de la classe IndiceNonValide sont utilisés dans la

classe Collection lorsque l'indice que l'on passe à une méthode se

trouve hors de la taille de la table.

Les constructeurs de la classe PrixNonValide sont utilisés dans la

classe Produit lorsque le prix d'un produit est mis à une valeur

inférieur à 5cts.

Dans le cadre de ce labo, on utilise que le constructeur const char\*

Mais nous avons decidé de garder quand même le constructeur

const string\*

Compilateur : MinGW-g++ 6.3.0

-----------------------------------------------------------------------------------

\*/

#ifndef INF\_2\_LABO\_4\_EXCEPTIONS\_H

#define INF\_2\_LABO\_4\_EXCEPTIONS\_H

#include <string>

#include <stdexcept> // logic\_error

**class** IndiceNonValide **:** **public** std**::**logic\_error **{**

**public:**

**explicit** IndiceNonValide**(const** std**::**string**&** s**);**

explicit IndiceNonValide(const char\* s);

};

class PrixNonValide : public std::logic\_error {

public:

explicit PrixNonValide(const std::string& s);

explicit PrixNonValide(const char\* s);

};

#endif //INF\_2\_LABO\_4\_EXCEPTIONS\_H

exceptions.cpp

/\*

-----------------------------------------------------------------------------------

Laboratoire : 04

Fichier : exception.cpp

Auteur(s) : Paul Reeve, Maxime Scharwath, Thibault Seem

Date : 20.04.2020

But : Implémentation des classes d'exception

Remarque(s) : -

Compilateur : MinGW-g++ 6.3.0

-----------------------------------------------------------------------------------

\*/

#include "exceptions.h"

**using** **namespace** std**;**

IndiceNonValide**::**IndiceNonValide**(const** std**::**string**&** s**)** **:** logic\_error**(**s**)** **{}**

IndiceNonValide**::**IndiceNonValide**(const** **char\*** s**)** **:** logic\_error**(**s**)** **{}**

PrixNonValide**::**PrixNonValide**(const** std**::**string**&** s**)** **:** logic\_error**(**s**)** **{}**

PrixNonValide**::**PrixNonValide**(const** **char\*** s**)** **:** logic\_error**(**s**)** **{}**

produit.h

/\*

-----------------------------------------------------------------------------------

Laboratoire : 04

Fichier : produit.h

Auteur(s) : Paul Reeve, Maxime Scharwath, Thibault Seem

Date : 20.04.2020

But : La classe Produit regroupe les fonctions suivantes :

- std::ostream& operator<<(std::ostream& lhs, const Produit& rhs) :

Surcharge de l'opérateur de flux "<<" pour afficher

les informations du produit sous la forme suivante :

(n°, "libellé", prix)

- bool operator==(const Produit& lhs, const Produit& rhs);

Opérateur de comparaison "==" afin de voir si deux produits sont

identiques.

- Produit(size\_t numero, std::string libelle, double prix) :

Un constructeur prenant 3 paramètres en entrée : un entier pour le

numéro du produit, un string pour le libbelé et un double pour

le prix. Si le prix reçu en paramètre est inférieur à 5cts,

une exception PrixNonValide est lancée (cf. exceptions.h)

- void setPrix(double nouveauPrix) : accesseur permettant de définir

le prix du produit. Si le prix est inférieur à 5cts, une exception

de type PrixNonValide est lancée (cf. exceptions.h).

- void majoration(double pourcent) : Une méthode permettant de

majorer le prix de l'objet d'un certain pourcent.

Remarque(s) : -

Compilateur : MinGW-g++ 6.3.0

-----------------------------------------------------------------------------------

\*/

#ifndef INF\_2\_LABO\_4\_PRODUIT\_H

#define INF\_2\_LABO\_4\_PRODUIT\_H

#include <string>

#include <iostream>

**class** Produit **{**

/\*\*

\* Surcharge de l'opérateur sur un flux "<<" afin d'afficher le numéro, le libellé

\* et le prix du produit

\*

\* **@param** lhs : Flux sur lequel on veut écrire

\* **@param** rhs : Produit qu'on souhaite afficher

\* **@return** : Flux sur lequel on écrit

\*/

**friend** std**::**ostream**&** **operator<<(**std**::**ostream**&** lhs**,** **const** Produit**&** rhs**);**

/\*\*

\* Opérateur de compraison " == " afin de tester si deux produits sont identiques

\*

\* @param lhs : Premier produit à comparer

\* @param rhs : Deuxième produit à comparer

\* @return : Résultat de la comparaison : renvoie true si les deux produits

\* sont identiques

\* renvoie false si les deux produits sont différents

\*/

friend bool operator==(const Produit& lhs, const Produit& rhs);

public:

/\*\*

\* Constructeur de la classe Produit

\*

\* @param numero : Numéro du produit créé

\* @param libelle : Libellé attaché au produit.

\* @throws : PrixNonValide::PrixNonValide si le prix est inférieur ou égal

\* à 5cts

\* @param prix : Prix du produit

\*/

Produit(size\_t numero, const std::string& libelle, double prix);

/\*\*

\* Méthode permettant de définir une nouvelle valeur pour le prix.

\*

\* @param nouveauPrix : Nouveau prix à pour le produit

\* @throws : PrixNonValide::PrixNonValide si le prix est inférieur

\* ou égal à 5cts

\*/

void setPrix(double nouveauPrix);

/\*\*

\* Méthode permettant de calculer une majoration sur le prix du produit

\*

\* @param pourcent : Pourcentage dont on veut majorer le prix

\*/

void majoration(double pourcent);

private:

size\_t numero;

std::string libelle;

double prix;

};

#endif //INF\_2\_LABO\_4\_PRODUIT\_H

produit.cpp

/\*

-----------------------------------------------------------------------------------

Laboratoire : 04

Fichier : produit.cpp

Auteur(s) : Paul Reeve, Maxime Scharwath, Thibault Seem

Date : 20.04.2020

But : Implémentation de la classe Produit, et de ses différentes fonctions.

Remarque(s) : -

Compilateur : MinGW-g++ 6.3.0

-----------------------------------------------------------------------------------

\*/

#include "produit.h"

#include "exceptions.h"

#include <iostream>

#include <iomanip>

**using** **namespace** std**;**

// Constructeur de la classe Produit

Produit**::**Produit**(size\_t** numero**,** **const** std**::**string**&** libelle**,** **double** prix**)** **{**

**if** **(**prix **<** 0.05**)** **{**

**throw** PrixNonValide**(**"Erreur dans Produit::Produit : "

"\nle prix doit etre >= 5 cts !"**);**

**}**

**this->**numero **=** numero**;**

**this->**libelle **=** libelle**;**

**this->**prix **=** prix**;**

**}**

// Surcharge de l'opérateur sur un flux "<<" afin d'afficher le numéro, le libellé

// et le prix du produit

std**::**ostream**&** **operator<<(**std**::**ostream**&** lhs**,** **const** Produit**&** rhs**)** **{**

lhs **<<** "(" **<<** rhs**.**numero **<<** ", \""

**<<** rhs**.**libelle **<<** "\", "

**<<** fixed **<<** setprecision**(**2**)** **<<** rhs**.**prix **<<** ")"**;**

**return** lhs**;**

**}**

//Opérateur de compraison " == " afin de tester si deux produits sont identiques

**bool** **operator==(const** Produit**&** lhs**,** **const** Produit**&** rhs**)** **{**

**return** lhs**.**numero **==** rhs**.**numero

**&&** lhs**.**libelle **==** rhs**.**libelle

**&&** lhs**.**prix **==** rhs**.**prix**;**

**}**

// Méthode permettant de définir une nouvelle valeur pour le prix.

**void** Produit**::**setPrix**(double** nouveauPrix**)** **{**

**if** **(**nouveauPrix **<** 0.05**)** **{**

**throw** PrixNonValide**(**"Erreur dans Produit::setPrix : "

"\nle prix doit etre >= 5 cts !");

}

prix = nouveauPrix;

}

// Méthode permettant de calculer une majoration sur le prix du produit

void Produit::majoration(double pourcent) {

prix += prix \* (pourcent / 100.0);

}

collection\_g.h

/\*

-----------------------------------------------------------------------------------

Laboratoire : 04

Fichier : collection\_g.h

Auteur(s) : Paul Reeve, Maxime Scharwath, Thibault Seem

Date : 20.04.2020

But : Déclaration et implémentation de la classe collection permettant

de stocker des élément génériques sous forme de table.

La classe Collection contient les méthodes suivantes :

- std::ostream &operator<<

(std::ostream &os, const Collection<T, Container> &c) :

Surcharge de l'opérateur d'écriture sur un flux "<<" afin de

pouvoir écrire une table générique sur le flux sous la forme :

[element1, element2, ...]

- void ajouter(const T &valeur) : Permet d'ajouter un élément de

type T à la table de l'objet de type Collection.

- size\_t taille() const : retourne la taille de la table contenue

dans l'objet de type Collection.

- T &get(size\_t index) : Retourne l'élément à l'emplacement indiqué

par size\_t index. Si l'index fournit en paramètre est supérieur à

la taille de la table, une exception de type IndiceNonValide est

lancée (cf. exception.h).

- bool contient(const T &valeur) const : Retourne true si l'élément

valeur est présent dans la table de l'objet de type Collection,

et true si l'élément n'est pas présent.

- void vider() : Supprime tous les éléments dans la table de l'objet.

- void parcourir(std::function<void(T&)> fn) : Applique la fonction

fn à tous les éléments de la table de l'objet.

Remarque(s) : -

Compilateur : MinGW-g++ 6.3.0

-----------------------------------------------------------------------------------

\*/

#ifndef INF\_2\_LABO\_4\_COLLECTION\_G\_H

#define INF\_2\_LABO\_4\_COLLECTION\_G\_H

#include <algorithm>

#include <iterator>

#include <functional>

#include <iostream>

#include "exceptions.h"

**template** **<typename** T**,** **template** **<typename,** **typename=**std**::**allocator**<**T**>>** **class** Container**>**

**class** Collection**;**

/\*\*

\* Surcharge de l'opérateur sur un flux "<<" afin d'afficher une collection

\*

\* **@param** os : Flux sur lequel on veut écrire

\* **@param** c : Collection qu'on souhaite afficher

\* @return : Flux sur lequel on écrit

\*/

template <typename T, template <typename, typename=std::allocator<T>> class Container>

std::ostream& operator<<(std::ostream& os, const Collection<T, Container>& c) {

os << '[';

for (auto i = c.data.begin(); i != c.data.end(); ++i) {

if (i != c.data.begin()) {

os << ", ";

}

os << \*i;

}

os << ']';

return os;

}

template <typename T, template <typename, typename=std::allocator<T>> class Container>

class Collection {

friend std::ostream& operator<< <>(

std::ostream& os,

const Collection<T, Container>& c

);

public:

/\*\*

\* Ajoute un objet de type T au bout de la Collection

\* @param valeur L'objet à ajouter

\*/

void ajouter(const T& valeur) {

data.push\_back(valeur);

}

/\*\*

\* Renvoie la taille de la Collection

\* @return taille de la

\*/

size\_t taille() const {

return (size\_t) distance(data.begin(), data.end());

}

/\*\*

\* Retourne une référence à l'élément de la Collection

\* étant à l'emplacement index

\* @param index l'index de l'élément voulu

\* @return référence à l'élément voulu

\*/

T& get(size\_t index) {

if (index >= taille()) {

throw IndiceNonValide(

"Erreur dans Collection::get : \n"

"n doit etre strictement plus petit que collection.size()"

);

}

auto it = data.begin();

for (size\_t i = 0; i != index && it != data.end(); ++i, ++it);

return \*it;

}

/\*\*

\* Renvoie vrai si l'élément cherché est dans la Collection, faux si non

\* @param valeur élément à chercher

\* @return true si trouvé, false si non

\*/

bool contient(const T& valeur) const {

return find(data.begin(), data.end(), valeur) != data.end();

}

/\*\*

\* Enlève tous les éléments de la Collection

\*/

void vider() {

data.clear();

}

/\*\*

\* Effectue une fonction sur chaque élément de la collection

\* @param fn la fonction à appliquer

\*/

void parcourir(std::function<void(T&)> fn) {

std::for\_each(data.begin(), data.end(), fn);

}

private:

Container<T> data;

};

#endif //INF\_2\_LABO\_4\_COLLECTION\_G\_H