main_labo_04.cpp

```
/*
 ------------------------------------------------------------------------------
 Laboratoire : 04
 Fichier     : main_labo_04.cpp
 Auteur(s)   : Nicolas Crausaz, Melvyn Herzig, Quentin Forestier
 Date        : 24.04.2020

 But         : Tester le fonctionnement d'un Produit.

               Tester de manipuler des Collections avec plusieurs types
               de données:
                  - char
                  - Produit
               Et différents conteneurs:
                  - Vector
                  - List

 Remarque(s) : /

 Compilateur : MinGW-g++ 6.3.0
 ------------------------------------------------------------------------------
 */

#include <cstdlib>         // std::EXIT_SUCCESS
#include <iostream>        // std::cout
#include <list>            // std::list
#include <vector>          // std::vector
#include "collection_g.h"  // Collection, operator<<
#include "exceptions.h"    // PrixNonValide, IndiceNonValide
#include "produit.h"       // Produit, SetPrix, GetPrix, operator=

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;
      {
         class Majoration
         {
         public:
            explicit Majoration (int pourcent): pourcent(pourcent) {};

            Produit& operator() (Produit& p)
            {
               double copiePrix = p.getPrix();
               p.setPrix((1.0 + pourcent / 100.0) * copiePrix);

               return p;
            }
         private:
            double pourcent;
         };

         // On parcourt la collection en majorant le prix de chacun
         // des produits de 10%
         c.parcourir(Majoration(10));
         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     : exceptions.h
 Auteur(s)   : Nicolas Crausaz, Melvyn Herzig, Quentin Forestier
 Date        : 24.04.2020

 But         : Implémenter deux classes exceptions pour:

               -  La vérification de prix.
               -  L'utilisation d'indices.

 Remarque(s) : Les deux classes héritent de std::logic_error

 Compilateur : MinGW-g++ 6.3.0
 ---------------------------------------------------------------------------
 */

#ifndef INF2_LABO4_EXCEPTIONS_H
#define INF2_LABO4_EXCEPTIONS_H

#include <stdexcept> // std::logic_error
#include <string>    // std::string

class IndiceNonValide : public std::logic_error
{
public:
   explicit IndiceNonValide (const std::string& s) : logic_error(s) {};
   explicit IndiceNonValide (const char* s)        : logic_error(s) {};
};

class PrixNonValide : public std::logic_error
{
public:
   explicit PrixNonValide (const std::string& s) : logic_error(s) {};
   explicit PrixNonValide (const char* s)        : logic_error(s) {};
};

#endif // INF2_LABO4_EXCEPTIONS_H
```

produit.h

```
/*
 ------------------------------------------------------------------------------
 Laboratoire : 04
 Fichier     : produit.h
 Auteur(s)   : Nicolas Crausaz, Melvyn Herzig, Quentin Forestier
 Date        : 24.04.2020

 But         : Implémenter une classe produit minimale
                   - Construction
                   - get/set du prix
                   - operator=
                   - operator<< (amitié)
                   - operator== (amitié)
                   - validation prix (amitié)

 Remarque(s) : /

 Compilateur : MinGW-g++ 6.3.0
 ------------------------------------------------------------------------------
 */

#ifndef INF2_LABO4_PRODUIT_H
#define INF2_LABO4_PRODUIT_H

#include <string> // std::string

class Produit
{
   friend void verificationPrix(double prix, const std::string& source);
   friend std::ostream& operator<<(std::ostream& os, const Produit& rhs);
   friend bool operator==(const Produit& lhs, const Produit& rhs);

public:
   Produit(unsigned no, const std::string& libelle, double prix);
   Produit(const Produit& produit) = default;
   double getPrix() const noexcept;
   void setPrix(double prix);
   Produit& operator=(Produit& rhs);

private:
   const unsigned no;
   const std::string libelle;
   double prix;
   static const double PRIX_MINIMUM;
};

#endif //INF2_LABO4_PRODUIT_H
```

produit.cpp

```
/*
 ------------------------------------------------------------------------------
 Laboratoire : 04
 Fichier     : produit.cpp
 Auteur(s)   : Nicolas Crausaz, Melvyn Herzig, Quentin Forestier
 Date        : 24.04.2020

 Compilateur : MinGW-g++ 6.3.0
 ------------------------------------------------------------------------------
 */

#include "produit.h"
#include <iostream>     // std::cout
#include <iomanip>      // std::fixed, std::setprecision
#include "exceptions.h" // PrixNonValide

// ------------------------
//    Membres statiques
// ------------------------

const double Produit::PRIX_MINIMUM = 0.05;

// ------------------------
//    Fonctions amies
// ------------------------

// Vérification du prix, lance une erreur PrixNonValide en cas d'erreur
void verificationPrix(double prix, const std::string& source)
{
   if (prix < Produit::PRIX_MINIMUM)
   {
      std::cout << source << std::endl;
      std::string errMessage = (std::string) "le prix doit etre >= " +
                               std::to_string((unsigned) (Produit::PRIX_MINIMUM * 100)) +
                               (std::string) + " cts !";

      throw PrixNonValide(errMessage);
   }
}

// Surcharge opérateur de flux <<
std::ostream& operator<<(std::ostream& os, const Produit& rhs)
{
   const short PRECISION_PRIX = 2;
   os << std::fixed << std::setprecision(PRECISION_PRIX);
   return os << "(" << rhs.no << ", \"" << rhs.libelle << "\", " << rhs.prix << ")";
}

// Opérateur d'égalité
bool operator==(const Produit& lhs, const Produit& rhs)
{
   return lhs.no == rhs.no && lhs.libelle == rhs.libelle && lhs.prix == rhs.prix;
}

// ------------------------
//    Méthodes publiques
// ------------------------

// Constructeur
Produit::Produit(unsigned no, const std::string& libelle, double prix)
   : no(no), libelle(libelle)
{
   try
   {
```

```
      verificationPrix(prix, "Erreur dans Produit::Produit : ");
      this->prix = prix;
   }
   catch (const PrixNonValide& e)
   {
      throw e;
   }
}

// Accesseurs
double Produit::getPrix() const noexcept
{
   return prix;
}

void Produit::setPrix(double prix)
{
   try
   {
      verificationPrix(prix, "Erreur dans Produit::setPrix : ");
      this->prix = prix;
   }
   catch (const PrixNonValide& e)
   {
      throw e;
   }
}

// Opérateur d'affectation
Produit& Produit::operator=(Produit& rhs)
{
   if (&rhs == this) return *this; // En cas d'auto-assignation

   (unsigned&) no = rhs.no;
   (std::string&) libelle = rhs.libelle;
   prix = rhs.prix;

   return *this;
}
```

collection_g.h

```
/*
 ---------------------------------------------------------------------------
 Laboratoire : 04
 Fichier     : collection_g.h
 Auteur(s)   : Nicolas Crausaz, Melvyn Herzig, Quentin Forestier
 Date        : 24.04.2020

 But         : Implémenter une classe permettant de stocker des objets
               quelconques avec une type de conteneur choisi.

                - Vérifier l'appartenance d'un objet
                - Savoir le nombre d'objet
                - Accéder/modifier un objet en position choisie
                - Une fonction qui applique une modification à tous les objets.

 Remarque(s) : La fonction contient() nécessite que l'opérateur ==
               soit implémenté pour le type T.
               La méthode vider nécessite l'implémentation de la méthode clear()
               pour le conteneur
               La méthode taille nécessite l'implémentation de la méthode size()
               pour le conteneur

 Compilateur : MinGW-g++ 6.3.0
 ---------------------------------------------------------------------------
 */
#ifndef INF2_LABO4_COLLECTION_G_H
#define INF2_LABO4_COLLECTION_G_H

template <typename T, template <typename, typename> class Conteneur>
class Collection;

template <typename T, template <typename, typename> class Conteneur>
std::ostream& operator<<(std::ostream& os, const Collection<T, Conteneur>& c);

template <typename T, template <typename, typename> class Conteneur>
const T& baseGet (const Collection<T, Conteneur>& collection, size_t index);

template <typename T, template <typename, typename> class Conteneur>
class Collection
{
   friend std::ostream& operator<<
           <T, Conteneur> (std::ostream& os, const Collection<T, Conteneur>& c);

   friend const T& baseGet
           <T, Conteneur> (const Collection<T, Conteneur>& collection, size_t index);

public:
   Collection() = default;

   explicit Collection(const Conteneur<T, std::allocator<T>>& c);

   void ajouter(const T& element);

   const T& get(size_t index) const;

   T& get(size_t index);

   bool contient(const T& element) const;

   void vider() noexcept;

   size_t taille() const noexcept;

   template <typename UnaryFunction>
```

```
   void parcourir(UnaryFunction function);

private:
   Conteneur<T, std::allocator<T>> conteneur;

};

#include "collectionImpl_g.h"

#endif // INF2_LABO4_COLLECTION_G_H
```

collectionImpl_g.h

```
/*
 ------------------------------------------------------------------------------
 Laboratoire : 04
 Fichier     : collectionImpl_g.h
 Auteur(s)   : Nicolas Crausaz, Melvyn Herzig, Quentin Forestier
 Date        : 24.04.2020

 Compilateur : MinGW-g++ 6.3.0
 ------------------------------------------------------------------------------
 */

#ifndef INF2_LABO4_COLLECTIONIMPL_G_H
#define INF2_LABO4_COLLECTIONIMPL_G_H

#include "collection_g.h"
#include <algorithm>    // std::find, std::for_each
#include <iterator>     // std::advance
#include "exceptions.h" // IndiceNonValide



// ------------------------
//    Fonctions amies
// ------------------------

// Opérateur de flux pour la collection
template <typename T, template <typename, typename> class Conteneur>
std::ostream& operator<<(std::ostream& os, const Collection<T, Conteneur>& c)
{
   os << "[";
   for (size_t i = 0; i < c.taille(); ++i)
   {
      if (i != 0)
      { os << ", "; }
      os << c.get(i);
   }
   return os << "]";
}

// Obtention d'un élément dans la conteneur, renvoi une lvalue
template <typename T, template <typename, typename> class Conteneur>
const T& baseGet(const Collection<T, Conteneur>& collection, size_t index)
{
   if (index >= collection.taille())
   {
      throw IndiceNonValide("n doit etre strictement plus petit que "
                            "collection.size()");
   }

   auto it = collection.conteneur.begin();
   std::advance(it, index);

   return *it;
}

// ------------------------
//    Méthodes publiques
// ------------------------

// Constructeur
template <typename T, template <typename, typename> class Conteneur>
Collection<T, Conteneur>::Collection(const Conteneur<T, std::allocator<T>>& c)
        : conteneur(c)
{}
```

```
// Ajout d'un élément dans le conteneur
template <typename T, template <typename, typename> class Conteneur>
void Collection<T, Conteneur>::ajouter(const T& element)
{
   conteneur.push_back(element);
}

// Obtention d'un élément dans la conteneur, renvoi une lvalue
template <typename T, template <typename, typename> class Conteneur>
T& Collection<T, Conteneur>::get(size_t index)
{
   try
   {
      return (T&) baseGet(*this, index);
   }
   catch (const IndiceNonValide& e)
   {
      std::cout << "Erreur dans Collection::get : " << std::endl;
      throw e;
   }
}

// Obtention d'un élément dans la conteneur, renvoi une rvalue
template <typename T, template <typename, typename> class Conteneur>
const T& Collection<T, Conteneur>::get(size_t index) const
{
   try
   {
      return baseGet(*this, index);
   }
   catch (const IndiceNonValide& e)
   {
      std::cout << "Erreur dans Collection::get : " << std::endl;
      throw e;
   }
}

// Verification qu'un element est dans le conteneur
template <typename T, template <typename, typename> class Conteneur>
bool Collection<T, Conteneur>::contient(const T& element) const
{
   return std::find(conteneur.begin(),
                    conteneur.end(),
                    element) != conteneur.end();
}

// Le conteneur doit implémenter la méthode clear()
template <typename T, template <typename, typename> class Conteneur>
void Collection<T, Conteneur>::vider() noexcept
{
   conteneur.clear();
}

// Le conteneur doit implémenter la méthode size()
template <typename T, template <typename, typename> class Conteneur>
size_t Collection<T, Conteneur>::taille() const noexcept
{
   return conteneur.size();
}

// Applique une fonction donnée pour chaque élément du conteneur
template <typename T, template <typename, typename> class Conteneur>
template <typename UnaryFunction>
void Collection<T, Conteneur>::parcourir(UnaryFunction function)
{
   std::transform(conteneur.begin(),
                  conteneur.end(),
```

```
                    conteneur.begin(),
                    function);
}

#endif // INF2_LABO4_COLLECTIONIMPL_G_H
```