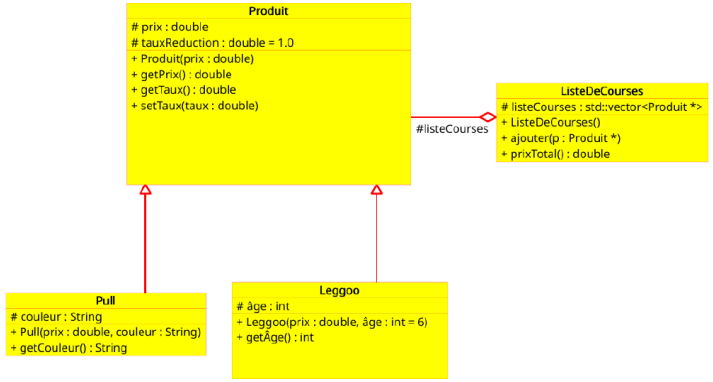
Université de Nice-Sophia Antipolis — POLYTECH
ELEC4 — 2020–2021

Examen de C++

Durée : 1h00
Aucun document autorisé
Mobiles interdits

Au vu de la situation Covid19, Amoozom promeut l'achat de cadeaux en ligne pour la nouvelle année, avec pour certains, des taux de réduction attractifs. Son catalogue propose des produits : des Pulls de couleurs diverses et des boîtes Leggoo variées pour tous les âges !

L'application à écrire comporte (dans un premier temps) 4 classes, données par le diagramme de classes suivant :



Dans ce diagramme, les # devant les variables indiquent qu'elles sont protégées et les + devant les méthodes indiquent qu'elles sont publiques. Enfin, les flèches entre les classes `Pull` et `Leggoo` d'une part, et `Produit` d'autre part, indiquent des liens d'héritage (relation *est-un*). Enfin, la flèche terminée par un losange indique une relation d'agrégation entre les classes `Produit` et `ListeDeCourses` : *i.e.* une liste de courses possède des produits (relation *a-un*).

Classes Produit, Pull et Leggo

▶ 1. Écrivez en C++ les classes `Produit`, `Pull` et `Leggoo` données par le diagramme précédent. Vous ajouterez la surcharge de l'opérateur `<<` pour écrire un produit (avec le taux de réduction, s'il y a lieu) dans un `std::ostream`. Pour cette dernière, attention à bien factoriser le code entre les différentes classes.

```
class Produit {
protected:
  double prix;
```



```
  double tauxReduction=1.0;

  std::string prixToString() const {
    std::ostringstream s;
    if (this->getTaux()!= 1.0)
      s << "soldé à ";
    s << this->getPrix() << " euros ";
    return s.str();
  }

public:
  Produit(const double p): prix(p) {}

  virtual ~Produit() {};
  double getPrix() const {
    return this->prix*getTaux();
  }
  double getTaux() const {
   return this->tauxReduction;
  }
  void setTaux(const double t) {
    assert(t>=0.0 && t<=1.0);
    this->tauxReduction=t;
  }
  // toString méthode abstraite!
  virtual std::string toString() const=0;

  friend std::ostream &operator<<(std::ostream &f, const Produit &p) {
    return f << p.toString();
  }
};
```

```
class Pull: public Produit {
protected:
  std::string couleur;
public:
  Pull(const double p, const std::string c): Produit(p), couleur(c) {}

  std::string getCouleur() const {
    return this->couleur;
  }

  virtual std::string toString() const override {
    std::ostringstream s;
    s << "Pull - " << this->couleur << " - ";
    s << Produit::prixToString ();
    return s.str();
  }
};
```

```
class Leggoo: public Produit {
protected:
  int age;
public:
  Leggoo(const double p, const int a=6) : Produit(p), age(a) {}
```

```
  int getAge() const {
    return this->age;
  }
  virtual std::string toString() const override {
    std::ostringstream s;
    s << "Leggoo - pour âges " << this->getAge() << "+ - ";
    s << Produit::prixToString();
    return s.str();
  }
};
```

Classe ListeDeCourses

▶ 2. Écrivez en C++ la classe `ListeDeCourses` donnée par le diagramme de classes précédent. La méthode `prixTotal` renvoie la somme des prix de tous les produits de la liste de courses. Vous écrirez également, son destructeur et la surcharge de l'opérateur `<<` pour écrire une liste de courses dans un `std::ostream`.

```
class ListeDeCourses {
private:
  std::vector<Produit *> listeCourses;
public:
  // le destucteur
  ~ListeDeCourses() {
    for (Produit *p: this->listeCourses)
      delete p;
  }
  void ajouter(Produit *p) {
    this->listeCourses.push_back(p);
  }
  double prixTotal() const {
    double total=0;
    for (Produit* p : this->listeCourses)
      total += p->getPrix();
    //
    return total;
  }
  std::string toString() const {
    std::ostringstream s;
    for (Produit* p : this->listeCourses)
      s << *p << "\n";
    s << "---------------------------------\n";
    s << "Total:" << this->prixTotal() << " euros\n";
    s <<"---------------------------------\n";
    return s.str();
  }
  friend std::ostream &operator<<(std::ostream &f,const ListeDeCourses &l) {
    return f<< l.toString();
  }
};
```


On veut ajouter à la classe `ListeDeCourses` la méthode `enregistrer` qui écrit la liste de courses courante dans un fichier (de type `std::ofstream`). Le nom du fichier (de type `std::string`) est passé en paramètre. Si une erreur se produit à l'ouverture du fichier, l'exception `FileException` avec un message d'erreur est émise.

▶ 3. Écrivez en C++ la classe `FileException` dans la laquelle vous redéfinirez la méthode `what()`.

```
class FileException: public std::exception {
private:
  std::string msg;
public:
  FileException(std::string s): msg(s) {};

  virtual const char*what() const noexcept override {
    return msg.c_str();
  }
};
```

▶ 4. Écrivez en C++ la méthode `enregister` qui écrit les `Produit` d'une `ListeDeCourses` courante dans un fichier. Si vous avez besoin de (re)définir d'autres méthodes, vous les écrirez en précisant bien les classes auxquelles elles appartiennent.

Dans la classe `ListeDeCourses` :

```
void enregistrer(const std::string &filename) const {
  std::ofstream oos(filename);
  if(!oos.is_open())
      throw FileException("Erreur:" + filename + "ouverture impossible");
  //ok écrire la liste
  for (Produit* p: this->listeCourses)
    oos << *p;
  //
  oos.close();
};
```

Dans la classe `Produit` :

```
virtual void ecrireProduit(std::ofstream &f) const =0;

friend std::ofstream&operator<<(std::ofstream&f,const Produit &p){
  p.ecrireProduit(f);
  return f;
}
```

Dans la classe `Pull` :

```
virtual void ecrireProduit(std::ofstream &f) const override {
    f.write((char*) this, sizeof(Pull));
```

```
}
```

Dans la classe `Leggoo` :

```
virtual void ecrireProduit(std::ofstream &f) const override {
  f.write((char*) this, sizeof(Leggoo));
}
```

.....

Classe de test

▶ 5. Écrivez en C++ la fonction `main` dans laquelle :

- vous créerez 2 listes de courses, une pour Mario et l'autre pour la Princesse Pêche avec les produits suivants :

  Liste de Mario :
  - Leggoo - Planet destroyer - âge 12 ans - 90 euros
  - Leggoo - Ma Petite Ferme - âge 6 ans - 70 euros
  - Pull - rouge - 40 euros

  Liste de Princesse Pêche :
  - Pull - magenta - 40 euros
  - Pull - rainbow - 55 euros
  - Leggo - My Happy Fish Tank - âge 8 ans - 10 euros

  Vous appliquerez une réduction de 50% sur le premier Leggoo de Mario, et une réduction de 30% sur son second Leggo. Pour Princesse Pêche, vous appliquerez une réduction de 50% sur le deuxième Pull.
- Vous afficherez les deux listes de courses avec, pour chacune, leur prix total.
- Vous enregistrez la liste de Mario dans un fichier de nom *"Mario"*. Vous ferez l'enregistrement dans un try-catch et, en cas d'erreur, vous afficherez sur la sortie d'erreur standard le message fourni par l'exception.

L'exécution de votre programme doit donner :

```
MarioListe :
Leggoo - pour âges 12+ - soldé à 45 euros
Leggoo - pour âges 6+ - soldé à 49 euros
Pull - rouge - 40 euros
---------------------------------
Total : 134 euros
---------------------------------
PêcheListe :
Pull - magenta - 40 euros
Pull - rainbow - soldé à 27.5 euros
Leggoo - pour âges 8+ - 10 euros
---------------------------------
Total : 77.5 euros
---------------------------------
```


Par exemple, l'ouverture erronée du fichier `/usr/Mario` produira :

```
Erreur : /usr/Mario ouverture impossible
```

```
int main() {
  ListeDeCourses MarioListe, PecheListe;
  //
  Produit *p = new Leggoo(90.0,12);
  p->setTaux(0.5);
  MarioListe.ajouter(p);
  p = new Leggoo(70.0);
  p->setTaux(0.7);
  MarioListe.ajouter(p);
  MarioListe.ajouter(new Pull(40.0,"rouge"));

  PecheListe.ajouter(new Pull(40,"magenta"));
  p = new Pull(55,"rainbow");
  p->setTaux(0.5);
  PecheListe.ajouter(p);
  PecheListe.ajouter(new Leggoo(10,8));
  //
  std::cout << "MarioListe : " << std::endl <<MarioListe;
  std::cout << "PêcheListe : " << std::endl <<PecheListe;
  //
  try {
    MarioListe.enregistrer("/usr/Mario");
  }
  catch(const FileException &e){
    std::cerr<<e.what()<<std::endl;
  }
  return EXIT_SUCCESS;
}
```

.....