```cpp
/*
-----------------------------------------------------------------------------
Nom du fichier  : main.cpp
Auteur(s)       : Alexandre Jaquier, Jonathan Friedli
Date creation   : 17.03.2022
Description     : Fichier principal du programme permettant de tester le bon
                  fonctionnement de la classe Squadron.
Compilateur     : Mingw-w64 g++ 8.1.0
-----------------------------------------------------------------------------
*/
#include <iostream>
#include "tieHunter.hpp"
#include "tieInterceptor.hpp"
#include "dreadnought.hpp"
#include "shuttle.hpp"
#include "squadron.hpp"

/**
 * Permet de tester les fonctionnalités liées aux vaisseaux
 */
void testShips() {
    std::cout << std::endl << "---------------------- Tests des Vaisseaux "
                              "----------------------" << std::endl << std::endl;
    std::cout << "Creation et affichage de vaisseaux" << std::endl;
    Shuttle shuttle(20);
    Dreadnought dreadnought(40);
    TieInterceptor tieInterceptor;
    TieHunter tieHunter;
    std::cout << shuttle << std::endl << std::endl <<
              dreadnought << std::endl << std::endl <<
              tieInterceptor << std::endl << std::endl <<
              tieHunter << std::endl << std::endl;
    std::cout << "Changement de nom pour Dreadnought :" << std::endl;
    dreadnought.setNickname("Dreadnought");
    std::cout << dreadnought << std::endl;

    std::cout << "Formule de consommation pour DreadNought avec une cargaison de "
                 "taille 40, la distance et la vitesse sont de 20 :" << std::endl;
    std::cout << dreadnought.getConsumption(20, 20) << std::endl;

    // Les méthodes non constantes ne peuvent pas être appelées sur une instance
    // constantes.
    std::cout << "Tests des ships constants :" << std::endl;
    const Dreadnought constantShip(40);
     //constantShip.setLoad(466);
     // impossible d'appeler cette fonction
    std::cout << constantShip << std::endl;

     std::cout << "Tests des transporters :" << std::endl;
    try {
        Shuttle shut(100);
    } catch (std::invalid_argument &e) {
        std::cout << e.what() << std::endl;
    }
    Dreadnought dread(155);
    try {
        dread.setLoad(100444444);
    } catch (std::invalid_argument &e) {
        std::cout << e.what() << std::endl;
    }
}
```

```
70  /**
71   * Permet de tester les fonctionnalités liées aux squadrons
72   */
73  void testSquadron() {
74
75      std::cout << std::endl << "--------------------- Tests des Squadrons "
76                                "----------------------" << std::endl << std::endl;
77
78      std::cout << "Creation d'un Squadron" << std::endl;
79      Squadron squad1("test1");
80      std::cout << squad1 << std::endl;
81
82      std::cout << "Creation d'un Squadron constant" << std::endl;
83
84
85      // Les méthodes non constantes ne peuvent pas être appelées sur une instance
86      // constantes.
87      const Squadron squadStat("testsConst");
88       // squadStat.addShipToSquadron()
89      std::cout << squadStat << std::endl;
90
91
92      std::cout << "Creation de vaisseaux" << std::endl;
93      Shuttle shuttle(20);
94      Dreadnought dreadnought(40);
95      TieInterceptor tieInterceptor;
96      TieHunter tieHunter;
97      std::cout << shuttle << std::endl << std::endl <<
98                dreadnought << std::endl << std::endl <<
99                tieInterceptor << std::endl << std::endl <<
100               tieHunter << std::endl << std::endl;
101
102     std::cout << "Ajout des vaisseaux dans le squadron1" << std::endl;
103     squad1.addShipToSquadron(shuttle);
104     squad1.addShipToSquadron(dreadnought);
105
106     std::cout << squad1 << std::endl;
107
108     std::cout << "Test du constructeur de copie (squad2)" << std::endl;
109     Squadron squad2(squad1);
110     std::cout << squad2 << std::endl;
111
112     squad1.addShipToSquadron(tieInterceptor);
113     squad1.addShipToSquadron(tieHunter);
114
115     std::cout << "Test de l'operateur d'affectation" << std::endl;
116     squad2 = squad1;
117     std::cout << "Squad 1" << std::endl << squad1 << std::endl;
118     std::cout << "Squad 2" << std::endl << squad2 << std::endl;
119
120     std::cout << "Test de l'operateur d'addition" << std::endl;
121     TieHunter add;
122     std::cout << "Squad 2 avec add" << std::endl << squad2 + add << std::endl;
123
124     std::cout << "Test de l'operateur de soustraction" << std::endl;
125     std::cout << "Squad 2 avec add" << std::endl << (squad2 - shuttle) <<
126     std::endl;
127
128     std::cout << "Test de la suppression des vaisseaux (shuttle est enlevé 2X)" <<
129               std::endl;
130     squad1.removeShipFromSquadron(shuttle);
131     squad1.removeShipFromSquadron(shuttle);
132     squad1.removeShipFromSquadron(dreadnought);
133     squad1.removeShipFromSquadron(tieInterceptor);
134     squad1.removeShipFromSquadron(tieHunter);
135     std::cout << squad1 << std::endl;
136
137
138
```

```
139       std::cout << "Choix du chef du squadron1 -> tieHunter" << std::endl;
140       squad1.setLeader(tieHunter);
141       std::cout << squad1 << std::endl;
142
143       std::cout << "Choix du chef du squadron1 avec un vaisseau qui n'en fait pas "
144                         "partie"   << std::endl;
145       TieHunter notMember;
146       squad1.setLeader(notMember);
147
148       std::cout << squad1 << std::endl;
149
150
151       std::cout << "Test de la formule de consommation (avec une distance et une "
152                    "vitesse de 100)" << std::endl;
153       std::cout << "Consommation de l'escadrille ne contenant qu'un TieHunter : "
154                 << squad1.getConsumption(100, 100) << std::endl;
155       std::cout << "Consommation d'un TieHunter : " << tieHunter.getConsumption(100,
156                                                                                 100)
157                 << std::endl;
158       std::cout << squad1 << std::endl << std::endl;
159
160       std::cout << "Affichage du vaisseau 0 via la methode get" << std::endl;
161       std::cout << squad1.getShip(0) << std::endl << std::endl;
162
163       std::cout << "Affichage d'un vaisseau en dehors des index limites via la "
164                    "methode get" << std::endl << std::endl;
165       try {
166           squad1.getShip(5);
167       } catch (std::out_of_range &e) {
168           std::cout << e.what() << std::endl;
169       }
170
171       std::cout << "Consommation d'un squadron avec une distance ou une vitesse "
172                    "invalide" << std::endl;
173       try {
174           squad1.getConsumption(-1, 10);
175       } catch (std::invalid_argument &e) {
176           std::cout << e.what() << std::endl;
177       }
178       try {
179           squad1.getConsumption(1, 1000000);
180       } catch (std::invalid_argument &e) {
181           std::cout << e.what() << std::endl;
182       }
183   }
184
185   int main() {
186       TieHunter blackLeader;
187       blackLeader.setNickname("Black leader");
188       TieHunter blackTwo;
189       Shuttle shuttle(23.4); // 23.4 tonnes de marchandises
190
191       Squadron squad("Black Squadron");
192       squad += blackLeader;
193       squad += blackTwo;
194       squad += shuttle;
195       squad.setLeader(blackLeader);
196       std::cout << squad << std::endl << std::endl;
197
198        // testShips();
199       // testSquadron();
200
201       return EXIT_SUCCESS;
202   }
203
204   /* --------------------------------------------------------------------------------*/
205
206
207
```

```cpp
#ifndef LABO1_SQUADRON_HPP
#define LABO1_SQUADRON_HPP

#include "ship.hpp"

class Squadron;

/**
 * Opérateur de flux de l'escadrille
 * @param os flux d'écriture
 * @param squadron escadrille à écrire
 * @return flux d'écriture
 */
std::ostream &operator<<(std::ostream &os, const Squadron &squadron);

/**
 * Opérateur plus de l'escadrille
 * @param squadron escadrille où ajouter un vaisseau
 * @param ship vaisseau à ajouter
 * @return une copie de l'escadrille avec le vaisseau ajouté
 */
Squadron operator+(const Squadron &squadron, Ship &ship);

/**
 * Opérateur moins de l'escadrille
 * @param squadron escadrille où enlever un vaisseau
 * @param ship vaisseau à enlever
 * @return une copie de l'escadrille avec le vaisseau enlevé
 */
Squadron operator-(const Squadron &squadron, const Ship &ship);

/**
 * Classe permettant de modéliser une escadrille contenant des
 * vaisseaux et potentiellement un chef.
 * @author Alexandre Jaquier
 * @author Jonathan Friedli
 */
class Squadron {
    struct Link {
        Ship *value;
        Link *next;
    };

public:
    /**
     * Constructeur de la classe Squadron
     * @param name nom de l'escadrille
     */
    explicit Squadron(const std::string &name);

    /**
    * Constructeur par copie de la classe Squadron
    * @param other escadrille à copier
    */
    Squadron(const Squadron &other);

    /**
    * Déstructeur de la classe Squadron
    */
    ~Squadron();

    /**
    * Opérateur d'affectation de l'escadrille, copie tous les vaisseaux ainsi que
    * le nom et le chef d'escadrille
    * @param other
    * @return
    */
    Squadron &operator=(const Squadron &other);

```

```
277       /**
278       * Méthode permettant de définir le chef de l'escadrille, si le vaisseau ne fait
279       * pas parti des membres de l'escadrille il y est ajouté
280       * @param newLeader chef de l'escadrille
281       */
282       void setLeader(Ship &newLeader);
283
284       /**
285        * Méthode permettant de rajouter un vaisseau dans l'escadrille
286        * @param ship
287        * @return une référence sur l'escadrille
288        */
289       Squadron &addShipToSquadron(Ship &ship);
290
291       /**
292        * Méthode permettant de retirer un vaisseau de l'escadrille
293        * @param ship
294        * @return une référence sur l'escadrille
295        */
296       Squadron &removeShipFromSquadron(const Ship &ship);
297
298       /**
299        * Méthode permettant de récupérer un vaisseau de l'escadrille
300        * @throws out_of_range si le paramètre est en dehors des index de la liste
301        * @param index index du vaisseau dans la liste
302        * @return une référence constante sur le vaisseau récupéré
303        */
304       const Ship &getShip(size_t index) const;
305
306       /**
307        * Méthode permettant de récupérer différentes informations sur l'escadrille
308        * @param speed vitesse de l'escadrille
309        * @param weight poids de l'escadrille
310        */
311       void squadronInfos(unsigned &speed, double &weight) const;
312
313       /**
314        * Méthode retournant une nouvelle escadrille contenant les vaisseaux de
315        * l'escadrille appelée et enlevant le vaisseau passé en paramètre
316        * @param ship vaisseau à enlever de l'escadrille
317        * @return une nouvelle escadrille
318        */
319       Squadron removeShip(const Ship &ship) const;
320
321       /**
322        * Méthode retournant une nouvelle escadrille contenant les vaisseaux de
323        * l'escadrille appelée et ajoutant le vaisseau passé en paramètre
324        * @param ship vaisseau à ajouter à l'escadrille
325        * @return une nouvelle escadrille
326        */
327       Squadron addShip(Ship &ship) const;
328
329       /**
330        * Opérateur permettant de rajouter un vaisseau dans l'escadrille
331        * @param ship vaisseau à ajouter
332        * @return une référence sur l'escadrille
333        */
334       Squadron &operator+=(Ship &ship);
335
336       /**
337        * Opérateur permettant de retirer un vaisseau de l'escadrille
338        * @param ship vaisseau à retirer
339        * @return une référence sur l'escadrille
340        */
341       Squadron &operator-=(const Ship &ship);
342
343
344
345
```

```
346       /**
347        * Méthode permettant de récupérer la consommation de l'escadrille
348        * @param distance distance parcourue
349        * @param speed vitesse de l'escadrille
350        * @return la consommation de l'escadrille
351        */
352       double getConsumption(double distance, unsigned speed) const;
353
354       /**
355        * Opérateur permettant de récupérer un vaisseau de l'escadrille
356        * @throws out_of_range si le paramètre est en dehors des index de la liste
357        * @param i index du vaisseau dans la liste
358        * @return une référence constante sur le vaisseau récupéré
359        */
360       const Ship &operator[](size_t i) const;
361
362       /**
363        * Permet d'afficher les informations d'une escadrille dans un flux
364        * @param os flux dans lequel nous écrivons
365        * @return une référence sur le flux modifié
366        */
367       std::ostream &toStream(std::ostream &os) const;
368
369   private:
370       /**
371        * Méthode permettant d'initialiser les paramètres de l'escadrille
372        */
373       void initVariables(const std::string &newName, Ship *newLeader, Ship *newHead);
374
375       Ship *leader;
376       Link *listHead;
377       std::string name;
378   };
379
380
381   #endif //LABO1_SQUADRON_HPP
382
383   /* ----------------------------------------------------------------------------------------*/
384
385   #include "squadron.hpp"
386   #include <cmath>
387
388   using namespace std;
389
390   Squadron::Squadron(const string &name) {
391       initVariables(name, nullptr, nullptr);
392   }
393
394   Squadron::Squadron(const Squadron &other) {
395       initVariables(other.name, other.leader, other.listHead->value);
396       Link *toCopy = other.listHead->next;
397       while (toCopy != nullptr) {
398           addShipToSquadron(*toCopy->value);
399           toCopy = toCopy->next;
400       }
401   }
402
403   Squadron::~Squadron() {
404       Link *tmp = listHead;
405       Link *toDelete = listHead;
406       while (tmp != nullptr) {
407           tmp = tmp->next;
408           delete toDelete;
409           toDelete = tmp;
410       }
411   }
412
413
414
```

```
415 void Squadron::initVariables(const string &newName, Ship *newLeader, Ship *newHead) {
416     this->name = newName;
417     this->leader = newLeader;
418     this->listHead = newHead == nullptr ? nullptr : new Link{newHead, nullptr};
419 }
420
421 Squadron &Squadron::operator=(const Squadron &other) {
422     if (this == &other) {
423         return *this;
424     }
425     Link *toDelete = listHead;
426     Link *tmp = toDelete->next;
427
428     while (toDelete != nullptr) {
429         delete toDelete;
430         toDelete = tmp;
431         if (tmp != nullptr) {
432             tmp = tmp->next;
433         }
434     }
435     initVariables(other.name, other.leader, other.listHead->value);
436     tmp = other.listHead->next;
437     while (tmp != nullptr) {
438         addShipToSquadron(*tmp->value);
439         tmp = tmp->next;
440     }
441     return *this;
442 }
443
444 void Squadron::setLeader(Ship &newLeader) {
445     addShipToSquadron(newLeader);
446     this->leader = &newLeader;
447 }
448
449 Squadron Squadron::addShip(Ship &ship) const {
450     Squadron newSquadron(*this);
451     return newSquadron.addShipToSquadron(ship);;
452 }
453
454 Squadron Squadron::removeShip(const Ship &ship) const {
455     Squadron newSquadron(*this);
456     return newSquadron.removeShipFromSquadron(ship);
457 }
458
459 Squadron &Squadron::addShipToSquadron(Ship &ship) {
460     Link *tmp = listHead;
461     if (tmp == nullptr) {
462         listHead = new Link{&ship, nullptr};
463         return *this;
464     }
465     while (tmp != nullptr) {
466         if (tmp->value == &ship) {
467             return *this;
468         }
469         if (tmp->next == nullptr) {
470             break;
471         }
472         tmp = tmp->next;
473     }
474     tmp->next = new Link{&ship, nullptr};
475     return *this;
476 }
477
478
479
480
481
482
483
```

```cpp
Squadron &Squadron::removeShipFromSquadron(const Ship &ship) {
    Link *toRemove = listHead;
    Link *tmp = listHead;
    while (toRemove != nullptr) {
        if (toRemove->value == &ship) {
            if (toRemove == listHead) {
                listHead = toRemove->next;
            } else {
                tmp->next = toRemove->next;
            }
            delete toRemove;
            break;
        }
        tmp = toRemove;
        toRemove = toRemove->next;
    }
    return *this;
}

const Ship &Squadron::getShip(size_t index) const {
    Link *tmp = listHead;
    size_t counter = 0;
    while (counter++ != index && tmp->next != nullptr) {
        tmp = tmp->next;
    }
    if (counter == index + 1) {
        return *tmp->value;
    }
    throw out_of_range("Le Squadron ne contient pas de vaisseau a cet index");
}

void Squadron::squadronInfos(unsigned &speed, double &weight) const {
    speed = 0;
    weight = 0;

    Link *tmp = listHead;
    while (tmp != nullptr) {
        if (speed > tmp->value->getMaxSpeed() || speed == 0) {
            speed = tmp->value->getMaxSpeed();
        }
        weight += tmp->value->getWeight();
        tmp = tmp->next;
    }
}

Squadron &Squadron::operator+=(Ship &ship) {
    return addShipToSquadron(ship);
}

Squadron &Squadron::operator-=(const Ship &ship) {
    return removeShipFromSquadron(ship);
}

const Ship &Squadron::operator[](size_t i) const {
    return getShip(i);
}


Squadron operator+(const Squadron &squadron, Ship &ship) {
    return squadron.addShip(ship);
}

Squadron operator-(const Squadron &squadron, const Ship &ship) {
    return squadron.removeShip(ship);
}
```

```
553  double Squadron::getConsumption(double distance, unsigned int speed) const {
554      unsigned maxSpeed;
555      double totalWeight;
556      squadronInfos(maxSpeed, totalWeight);
557      if (speed > maxSpeed || distance < 0) {
558          throw std::invalid_argument("Ce squadron ne peut pas atteindre une telle "
559                                      "vitesse ou une telle distance");
560      }
561
562      Link *ship = this->listHead;
563      double consumption = 0;
564
565      while (ship != nullptr) {
566          consumption += ship->value->getConsumption(distance, speed);
567          ship = ship->next;
568      }
569      return consumption;
570  }
571
572  ostream &operator<<(ostream &os, const Squadron &squadron) {
573      return squadron.toStream(os);
574  }
575
576  std::ostream &Squadron::toStream(std::ostream &os) const{
577      unsigned maxSpeed;
578      double squadronWeight;
579      squadronInfos(maxSpeed, squadronWeight);
580
581      os << "Squadron: " << name << endl;
582      os << " max speed: " << maxSpeed << " MGLT" << endl;
583      os << " total weight: " << squadronWeight << " tons" << endl;
584
585      os << endl << "-- Leader" << endl;
586      Squadron::Link *member = listHead;
587      if (leader != nullptr)
588          os << *leader << endl << endl;
589
590      os << "-- Members" << endl;
591      while (member != nullptr) {
592          if (member->value != leader)
593              os << *member->value << endl << endl;
594          member = member->next;
595      }
596      return os;
597  }
598
599  /* ----------------------------------------------------------------------------------------*/
600
601
602  #ifndef SHIP_HPP
603  #define SHIP_HPP
604
605  #include <ostream>
606  #include "shipCharacteristic.hpp"
607
608  class Ship;
609
610  /**
611   * Surcharge de l'opérateur d'écriture dans un flux afin d'y écrire les informations
612   * d'un vaisseau. Fait appel à la
613   * méthode toStream
614   * @param os flux dans lequel on écrit
615   * @param ship Vaisseaux dont nous allons afficher les informations
616   * @return une référence sur l'opérateur de flux
617   */
618  std::ostream &operator<<(std::ostream &os, const Ship &ship);
619
620
621
```

```cpp
/**
 * Classe modélisant toutes sorte de vaisseaux spatiaux.
 * @authors Alexandre Jaquier et Jonathan Friedli
 * @date 17.03.2022
 */
class Ship {

protected :
    /**
     * Constructeur de la classe vaisseau
     * @param characteristic pointeur sur les caractéristique du vaisseau telles
     * que la vitesse maximum ou le modèle
     */
    explicit Ship(ShipCharacteristic *characteristic);

public:
    /**
     * Destructeur de la classe vaisseau
     */
    virtual ~Ship() = default;

    /**
     * Permet d'afficher les informations d'un vaisseau dans un flux
     * @param os flux dans lequel nous écrivons
     * @return une référence sur le flux modifié
     */
    virtual std::ostream &toStream(std::ostream &os) const;

    /**
     * Permet de modifier le nickname du vaisseau
     * @param name nouveau nom
     */
    void setNickname(const std::string &name);

    /**
     * Renvoie le poids du vaisseau en tonne
     * @return le poids en tonne
     */
    virtual double getWeight() const;

    /**
     * Renvoie la vitesse maximale du vaisseau
     * @return la vitesse en MGLT
     */
    unsigned int getMaxSpeed() const;

    /**
     * Calcule la consommation d'un vaisseau compte tenu de sa vitesse, de la distance du trajet
     * et de son chargement
     * @throws invalid_argument si la vitesse voulue n'est pas atteignable par le
     * vaisseau ou la distance est négative
     * @param distance Distance parcourue en millions de km
     * @param speed Vitesse à laquelle le vaisseau avance en MGLT
     * @return La consomation en tonne
     */
    double getConsumption(double distance, unsigned speed) const;

private:
    /**
     * Construit un string représentant l'identité du vaisseau sous la forme "[<modèle> #<id>]"
     * @return l'identité du vaisseau
     */
    std::string getIdentity() const;

    unsigned int id;
    std::string nickname;
    const ShipCharacteristic *characteristic;
};
#endif /* SHIP_HPP */
```

```
691
692 /* ------------------------------------------------------------------------------------*/
693 #include "ship.hpp"
694 #include <cmath>
695 #include <iomanip>
696
697 Ship::Ship(ShipCharacteristic *characteristic) :
698         characteristic(characteristic) {
699     id = characteristic->nextId();
700 }
701
702 std::ostream &operator<<(std::ostream &os, const Ship &ship) {
703     return ship.toStream(os);
704 }
705
706 std::ostream &Ship::toStream(std::ostream &os) const {
707     return os << (nickname.empty() ? "" : (nickname + " ")) << (getIdentity()) + "\n"
708               << "  weight : "
709               << std::fixed << std::setprecision(2) << getWeight()
710               << " tons\n  max speed : " << characteristic->getMaxSpeed()
711               << " MGLT";
712 }
713
714 double Ship::getConsumption(double distance, unsigned speed) const {
715     if (speed > characteristic->getMaxSpeed() || distance < 0) {
716         throw std::invalid_argument("La vitesse ou la distance n'est pas "
717                                     "atteignable");
718     }
719     return cbrt(getWeight()) / 2 * log10(getWeight() * speed) * log10(distance + 1);
720 }
721
722 void Ship::setNickname(const std::string &name) {
723     nickname = name;
724 }
725
726 std::string Ship::getIdentity() const {
727     return "[" + characteristic->getModele() + " #" + std::to_string
728             (id) + "]";
729 }
730
731 double Ship::getWeight() const {
732     return characteristic->getWeight();
733 }
734
735 unsigned int Ship::getMaxSpeed() const {
736     return characteristic->getMaxSpeed();
737 }
738
739 /* ------------------------------------------------------------------------------------*/
740
741 #ifndef LABO1_TRANSPORTER_HPP
742 #define LABO1_TRANSPORTER_HPP
743
744 #include "ship.hpp"
745 #include "transporterCharacteristic.hpp"
746
747 /**
748  * Déclaration de la classe Transporter héritant de Ship et modélisant des vaisseaux transportant
749  * des cargaisons.
750  * @authors Alexandre Jaquier et Jonathan Friedli
751  * @date 17.03.2022
752  */
753 class Transporter : public Ship {
754 protected:
755
756
757
758
759
```

```
760      /**
761       * Constructeur de la classe Transporter
762       * @param currentLoad poids de la cargaison en tonne
763       * @param characteristic Pointeur sur les caractéristique d'un vaisseau pouvant
764       * transporter une cargaison
765       */
766      Transporter(double currentLoad, TransporterCharacteristic *characteristic);
767
768  public:
769      /**
770       * Renvoie le poids de la cargaison courrante en tonne
771       * @return le poids de la cargaison en tonne
772       */
773      double getLoad() const;
774
775      /**
776       * Permet de mettre à jours le poids de la cargaison
777       * @param load nouveau poids en tonne
778       */
779      void setLoad(double load);
780
781      /**
782       * Redéfinition de la fonction toStream. Elle affiche les informations du vaisseau dans un flux
783       * @param os flux d'écriture dans lequel on écrit
784       * @return Une référence sur le flux modifié
785       */
786      std::ostream &toStream(std::ostream &os) const override;
787
788      /**
789       * Renvoie le poids du vaisseau + celui de sa cargaison en tonne
790       * @return le poids total en tonne
791       */
792      double getWeight() const override;
793
794  private:
795      double currentLoad;
796      TransporterCharacteristic *characteristic;
797  };
798
799
800  #endif //LABO1_TRANSPORTER_HPP
801
802  /* ----------------------------------------------------------------------------------------*/
803
804
805  #include "transporter.hpp"
806  #include <iomanip>
807  #include <iostream>
808
809  Transporter::Transporter(
810          double currentLoad, TransporterCharacteristic *characteristic) :
811          Ship(characteristic) {
812      this->characteristic = characteristic;
813      setLoad(currentLoad);
814  }
815
816  double Transporter::getLoad() const {
817      return currentLoad;
818  }
819
820  void Transporter::setLoad(double load) {
821      if (load > this->characteristic->getMaxLoad() || load < 0) {
822          throw std::invalid_argument("Vous tentez de mettre une cargaison trop lourde"
823                                      " dans ce vaisseau !");
824      }
825      currentLoad = load;
826  }
827
828
```

```
std::ostream &Transporter::toStream(std::ostream &os) const {
    return Ship::toStream(os) << std::endl <<"  cargo : " << std::setprecision(1)
                              << currentLoad << " tons (max : "
                              << characteristic->getMaxLoad() << ")";
}

double Transporter::getWeight() const {
    return characteristic->getWeight() + currentLoad;
}

/* ------------------------------------------------------------------------------------------*/

#ifndef LABO1_TIEHUNTER_HPP
#define LABO1_TIEHUNTER_HPP

#include "ship.hpp"

/**
 * Déclaration de la classe TieHunter héritant de Vaisseau.
 * @authors Alexandre Jaquier et Jonathan Friedli
 * @date 17.03.2022
 */
class TieHunter : public Ship {
public:

    /**
     * Constructeur de la classe TieHunter
     */
    TieHunter();

private:
    static ShipCharacteristic characteristic;
};


#endif //LABO1_TIEHUNTER_HPP

/* ------------------------------------------------------------------------------------------*/

#include "tieHunter.hpp"

ShipCharacteristic TieHunter::characteristic(
        "TIE/LN", 100, 6
);

TieHunter::TieHunter() : Ship(&characteristic) {}

/* ------------------------------------------------------------------------------------------*/

#ifndef LABO1_TIEINTERCEPTOR_HPP
#define LABO1_TIEINTERCEPTOR_HPP

#include "ship.hpp"

/**
 * Déclaration de la classe TieInterceptor héritant de Vaisseau.
 * @authors Alexandre Jaquier et Jonathan Friedli
 * @date 17.03.2022
 */
class TieInterceptor : public Ship {
public:
    /**
     * Constructeur de la classe TieInterceptor
     */
    TieInterceptor();
private:
    static ShipCharacteristic characteristic;
};
#endif //LABO1_TIEINTERCEPTOR_HPP
```

```
898
899 /* ------------------------------------------------------------------------------------*/
900
901 #include "tieInterceptor.hpp"
902
903 ShipCharacteristic TieInterceptor::characteristic(
904         "TIE/IN", 110, 5
905 );
906
907 TieInterceptor::TieInterceptor() : Ship(&characteristic) {}
908
909 /* ------------------------------------------------------------------------------------*/
910
911 #ifndef LABO1_DREADNOUGHT_HPP
912 #define LABO1_DREADNOUGHT_HPP
913
914 #include "transporter.hpp"
915
916
917 /**
918  * Déclaration de la classe Dreadnought héritant de Transporter.
919  * @authors Alexandre Jaquier et Jonathan Friedli
920  * @date 17.03.2022
921  */
922 class Dreadnought : public Transporter {
923 public:
924
925     /**
926      * Constructeur de la classe Dreadnought
927      * @param load poids de la cargaison en tonne
928      */
929     explicit Dreadnought(double load);
930
931 private:
932     static TransporterCharacteristic characteristic;
933 };
934
935
936 #endif //LABO1_DREADNOUGHT_HPP
937
938 /* ------------------------------------------------------------------------------------*/
939
940 #include "dreadnought.hpp"
941
942 TransporterCharacteristic Dreadnought::characteristic(
943         "Super-class Star Destroyer", 40, 9000000000, 250000
944 );
945
946 Dreadnought::Dreadnought(double load) : Transporter(
947         load, &characteristic
948 ) {}
949
950 /* ------------------------------------------------------------------------------------*/
951
952 #ifndef LABO1_SHUTTLE_HPP
953 #define LABO1_SHUTTLE_HPP
954
955 #include "transporter.hpp"
956 #include "transporterCharacteristic.hpp"
957
958 /**
959  * Déclaration de la classe shuttle héritant de Transporter.
960  * @authors Alexandre Jaquier et Jonathan Friedli
961  * @date 17.03.2022
962  */
963 class Shuttle : public Transporter {
964 public:
965
966
```

```
967       /**
968        * Constructeur de la classe Shuttle
969        * @param load poids de la cargaison en tonne
970        */
971       explicit Shuttle(double load);
972
973   private:
974       static TransporterCharacteristic characteristic;
975   };
976
977
978   #endif //LABO1_SHUTTLE_HPP
979
980   /* ------------------------------------------------------------------------------------*/
981
982   #include "shuttle.hpp"
983
984   TransporterCharacteristic Shuttle::characteristic(
985           "Lambda-class shuttle", 54, 360, 80
986   );
987
988   Shuttle::Shuttle(double load) : Transporter(
989           load, &characteristic
990   ) {}
991
992   /* ------------------------------------------------------------------------------------*/
993
994   #ifndef LABO1_SHIPCHARACTERISTIC_HPP
995   #define LABO1_SHIPCHARACTERISTIC_HPP
996
997
998   #include <string>
999
1000  /**
1001   * Classe modélisant les différentes caractéristiques d'un vaisseau spatial telle
1002   * que sa vitesse maximale, son modèle et son poids.
1003   * @authors Alexandre Jaquier et Jonathan Friedli
1004   * @date 17.03.2022
1005   */
1006  class ShipCharacteristic {
1007  public:
1008      /**
1009       * Constructeur de la classe ShipCharacteristic
1010       * @param modele modele du vaisseau
1011       * @param maxSpeed Vitesse maximale en MGLT
1012       * @param weight poids du vaisseau en tonne
1013       */
1014      ShipCharacteristic(const std::string &modele, unsigned maxSpeed, double weight);
1015
1016      /**
1017       * Destructeur de la classe ShipCharacteristic
1018       */
1019      virtual ~ShipCharacteristic() = default;
1020
1021      /**
1022       * Renvoie le modèle du vaisseau
1023       * @return le modèle
1024       */
1025      const std::string &getModele() const;
1026
1027      /**
1028       * Renvoie la vitesse max du vaisseau en MGLT
1029       * @return la vitesse max en MGLT
1030       */
1031      unsigned int getMaxSpeed() const;
1032
1033
1034
1035
```

```
    /**
     * Renvoie le poids du vaisseau en tonne
     * @return le poids en tonne
     */
    double getWeight() const;



    /**
     * Permet de générer un id dépendant du type du vaisseau
     * @return le prochain id utilisable
     */
    unsigned int nextId();


private:
    const std::string modele;
    const unsigned maxSpeed;
    const double weight;
    unsigned int counter;
};


#endif //LABO1_SHIPCHARACTERISTIC_HPP


/* ---------------------------------------------------------------------------------------*/

#include "shipCharacteristic.hpp"

ShipCharacteristic::ShipCharacteristic(const std::string &modele, unsigned maxSpeed,
                                       double weight) : modele(modele), maxSpeed
        (maxSpeed), weight(weight), counter(1) {}

const std::string &ShipCharacteristic::getModele() const {
    return modele;
}

unsigned int ShipCharacteristic::getMaxSpeed() const {
    return maxSpeed;
}

double ShipCharacteristic::getWeight() const {
    return weight;
}

unsigned int ShipCharacteristic::nextId() {
    return counter++;
}

/* ---------------------------------------------------------------------------------------*/


#ifndef LABO1_TRANSPORTERCHARACTERISTIC_HPP
#define LABO1_TRANSPORTERCHARACTERISTIC_HPP


#include "shipCharacteristic.hpp"

/**
 * Classe héritant de shipCharacteristic permettant de modéliser le poids maximal de
 * la cargaison d'un vaisseau
 */
class TransporterCharacteristic : public ShipCharacteristic {
public:
```

```
1105        /**
1106         * Constructeur de la classe TransporterCharacteristic
1107         * @param modele Modèle du vaisseau
1108         * @param maxSpeed Vitesse max du vaisseau en MGLT
1109         * @param weight Poids du vaisseau en tonne
1110         * @param maxLoad Poids maximal de la cargaison en tonne
1111         */
1112        TransporterCharacteristic(const std::string& modele, unsigned maxSpeed,
1113                                  double weight, double maxLoad);
1114
1115
1116
1117        /**
1118         * Renvoie le poids maximal de la cargaison en tonne
1119         * @return le poids max en tonne
1120         */
1121        double getMaxLoad() const;
1122
1123    private:
1124        double maxLoad;
1125    };
1126
1127
1128    #endif //LABO1_TRANSPORTERCHARACTERISTIC_HPP
1129
1130
1131    /* ---------------------------------------------------------------------------------------*/
1132
1133
1134    #include "transporterCharacteristic.hpp"
1135
1136    TransporterCharacteristic::TransporterCharacteristic(
1137            const std::string &modele, unsigned maxSpeed, double weight, double maxLoad)
1138            : ShipCharacteristic(modele, maxSpeed, weight), maxLoad(maxLoad) {}
1139
1140    double TransporterCharacteristic::getMaxLoad() const {
1141        return maxLoad;
1142    }
1143
```