```cpp
#include "Headers/Destroyer.hpp"
#include <cmath>

unsigned int Destroyer::serialNumberCounter = 0;

unsigned int Destroyer::getNextSerialNumber() {
    return ++serialNumberCounter;
}

Destroyer::Destroyer(double currentCapacity) : CargoShip(getNextSerialNumber(), currentCapacity) {
    if(currentCapacity < 0 || currentCapacity > this->getMaxCapacity()) {
     throw std::invalid_argument("Capacity must be greater than 0 and less than or equal to max capacity");
    }
}

double Destroyer::getMaxCapacity() const {
    return 250 * pow(10, 3);
}

double Destroyer::getModelWeight() const {
    return 9 * pow(10, 9);
}

std::string Destroyer::getModel() const {
    return "Super-class Star Destroyer";
}

unsigned int Destroyer::getModelSpeedMax() const {
    return 40;
}

```

```cpp
#ifndef POA_L2_SHUTTLE_HPP
#define POA_L2_SHUTTLE_HPP

#include "CargoShip.hpp"

/**
 * Represent a specific model of a Star Wars spaceship cargo
 * @link CargoShip
 * @version 1.0
 * @author Nelson Jeanrenaud
 * @author André Marques Nora
 */
class Shuttle : public CargoShip {
   /// Count of the number of ships created
   static unsigned int serialNumberCounter;
   unsigned int getNextSerialNumber() override;
public:
   /**
    * Construct a shuttle
    * @param currentCapacity weight of the load
    * @throw invalid_argument exception
    */
   explicit Shuttle(double currentCapacity);
   /** Get max capacity of the ship in tons
   * @return max capacity of the ship in tons
   */
   double getMaxCapacity() const override;
   /**
   * Get the weight of the ship in tons
   * @return the weight of the ship in tons
   */
   double getModelWeight() const override;
   /**
   * Get the model of the ship
   * @return the model of the ship
   */
   std::string getModel() const override;
   /**
   * Get the maximum speed of the ship model
   * @return the maximum speed of the ship model
   */
   unsigned int getModelSpeedMax() const override;
};


#endif //POA_L2_SHUTTLE_HPP

```

```hpp
#ifndef POA_L2_TIEIN_HPP
#define POA_L2_TIEIN_HPP

#include "Ship.hpp"
/**
 * @brief Represent a specific model of a Star Wars spaceship
 * @link Ship
 * @version 1.0
 * @author Nelson Jeanrenaud
 * @author André Marques Nora
 */
class TieIn : public Ship {
    /// Count of the number of ships created
    static unsigned int serialNumberCounter;
    /**
     * @brief Get the Next Serial Number object. Increments the counter
     *
     * @return the next serial number
     */
    unsigned int getNextSerialNumber() override;
public:
    /**
     * @brief Construct a new TieIn object
     */
    TieIn();
    std::string getModel() const override;
    unsigned int getModelSpeedMax() const override;
    double getModelWeight() const override;
};


#endif //POA_L2_TIEIN_HPP

```

```cpp
#include "Headers/CargoShip.hpp"
#include <iomanip>

CargoShip::CargoShip(unsigned int serialNumber, double currentCapacity) : currentCapacity(currentCapacity), Ship(serialNumber) {}

double CargoShip::getCurrentCapacity() const {
    return currentCapacity;
}

std::ostream& CargoShip::toStream(std::ostream &out) const {
    Ship::toStream(out);
    out << std::fixed << std::setprecision(1) << "cargo : " << currentCapacity << " tons (max : "
    << getMaxCapacity() << ")\n";
    return out;
}

```

```cpp
#ifndef POA_L2_SHIP_H
#define POA_L2_SHIP_H

#include "iostream"

class Ship;

std::ostream& operator<<(std::ostream& out, const Ship& ship);


/**
 * Represent a Star Wars spaceship
 * @version 1.0
 * @author Nelson Jeanrenaud
 * @author André Marques Nora
 */
class Ship {
    /// nickname given to this spaceship, can be null
    std::string nickname;
    /// serial number of this ship
    unsigned int serialNumber;
protected:
    /**
     * Construct a new Ship object
     *
     * @param serialNumber serial number of this ship
     */
    explicit Ship(unsigned int serialNumber);

public:
    virtual ~Ship();
    friend std::ostream& operator<<(std::ostream& out, const Ship&
ship);
    virtual std::ostream& toStream(std::ostream& out) const;
    /**
     * Give a new nickname to this ship
     *
     * @param newNickname nickname given to this ship
     */
    void setNickname(const std::string &newNickname);
    /**
     * Get the nickname of the ship
     * @return the nickname of the ship, if the ship has no nickname,
returns an empty string
     */
    std::string getNickname() const;
    /**
     * Get the consumption of the ship in tons
     * @param distance distance traveled in mio km
     * @param speed speed of the ship in MGLT
     * @return the consumption of the ship in tons
     */
    double getConsumption(unsigned distance, unsigned speed) const;
```

```
52      /**
53       * Get the Model object
54       *
55       * @return the model of the ship
56       */
57      virtual std::string getModel() const = 0;
58      /**
59       * Get the maximum speed of the ship model
60       *
61       * @return the maximum speed of the ship model
62       */
63      virtual unsigned int getModelSpeedMax() const = 0;
64      /**
65       * Get the weight of the ship model
66       *
67       * @return the weight of the ship model
68       */
69      virtual double getModelWeight() const = 0;
70      /**
71       *  Get the serial number of the next ship to be built
72       *
73       * @return the serial number of the next ship to be built
74       */
75      virtual unsigned int getNextSerialNumber() = 0;
76  };
77
78
79  #endif //POA_L2_SHIP_H
80
```

```hpp
#ifndef POA_L2_SQUADRON_HPP
#define POA_L2_SQUADRON_HPP

#include <vector>
#include "Ship.hpp"

/**
* Representation of a squadron of ships
* @link Ship
* @version 1.0
* @author Nelson Jeanrenaud
* @author André Marques Nora
*
*/
class Squadron {
   /**
   * Member of the squadron, node of the linked list
   */
   class Member {
     /**
      * Pointer to the ship of the member
      */
     const Ship* ship;
     /**
      * Pointer to the next member of the squadron
      */
     Member* next;
   public:
     /**
      * Construct a new Member object
      * @param ship the ship of the member
      */
     explicit Member(const Ship* ship);
     /**
      * Get the ship of the member
      * @return the ship of the member
      */
     const Ship& getShip() const;
     /**
      * Get the next member of the squadron
      * @return the next member of the squadron
      */
     Member* getNext() const;
     /**
      * Set the next member of the squadron
      * @param next the next member of the squadron
      */
     void setNext(Member* next);

     /**
      * Checks if the member has a next member
      * @return true if the member has a next member, false otherwise
      */
```

```
 54      bool hasNext() const;
 55    };
 56
 57    /// Ships contained in the squadron
 58    Member *firstMember;
 59
 60    /// Leader of the squadron
 61    Member *leader;
 62
 63    /// Name of the squadron
 64    std::string name;
 65
 66    /**
 67    * Get the member of the squadron that has the ship passed as
    parameter.
 68    * @param parameter-ship the ship to search
 69    * @return the member of the squadron that has the ship passed as
    parameter.
 70    */
 71    Member* getMember(const Ship& ship) const;
 72    /**
 73    * Get the member of the squadron at the given index
 74    *
 75    * @param index the index of the member to get
 76    * @return Member* the member of the squadron at the given index
 77    * @throw invalid_argument exception
 78    */
 79    Member* getMember(unsigned index) const;
 80    /**
 81    * Checks if the squadron is empty.
 82    * @return true if the squadron is empty, false otherwise
 83    */
 84    bool isEmpty() const;
 85    /**
 86    * Empty the squadron
 87    * @param squadron the squadron to copy
 88    */
 89    void emptySquad();
 90    /**
 91     * Copy a squadron
 92     * @param squadron squadron to copy
 93     */
 94    void copySquad(const Squadron& squadron);
 95 public:
 96
 97    /**
 98    * Construct a new Squadron object
 99    * @param name the name of the squadron
100    */
101    explicit Squadron(const std::string &name);
102    /**
103    * Construct a new Squadron object by copying the given squadron
104    *
```

```
105     * @param squadron the squadron to copy
106     */
107     Squadron(const Squadron& squadron);
108     /**
109      * operator of affectation
110      * @param squadron
111      * @return Squadron
112      */
113     Squadron& operator=(const Squadron& squadron);
114     /**
115     * Destroys the Squadron object
116     */
117     ~Squadron();
118 
119     /** Writes the details of the squadron into the output stream
120     * @param out output stream
121     * @param squadron squadron to be written
122     * @return output stream
123     */
124     friend std::ostream& operator<<(std::ostream& out, const Squadron
    & squadron);
125 
126     /**
127     * Set leader of the squadron
128     * @param ship the ship to set as leader
129     * @throw invalid_argument exception
130     */
131     void setLeader(const Ship& ship);
132     /**
133     * Remove leader of the squadron
134     */
135     void removeLeader();
136     /**
137     * Get leader of the squadron
138     * @return the leader of the squadron
139     */
140     const Ship* getLeader() const;
141     /**
142      * Set name of squadron
143      * @param name squadron's name
144      */
145     void setName(std::string name);
146     /**
147      * Get name of squadron
148      * @return string squadron's name
149      */
150     std::string getName() const;
151     /**
152     * Writes the details of the squadron into the output stream.
153     * @param out output stream
154     * @return output stream
155     */
156     std::ostream& toStream(std::ostream &out) const;
```

```
157     /**
158     * Adds a ship to the squadron
159     * @param ship the ship to add
160     * @throw invalid_argument exception
161     */
162     void addShipToSelf(const Ship& ship);
163     /**
164     * Adds a ship to a copy of the squadron
165     * @param ship the ship to add
166     * @return the copy of the squadron with the ship added
167     */
168     Squadron addShip(const Ship& ship) const;
169     /**
170     * Removes a ship from the squadron
171     * @param ship the ship to remove
172     * @throw invalid_argument exception
173     */
174     void removeShipToSelf(const Ship& ship);
175     /**
176     * Removes a ship from a copy of the squadron
177     * @param ship the ship to remove
178     * @return the copy of the squadron with the ship removed
179     */
180     Squadron removeShip(const Ship& ship) const;
181     /**
182     * Adds a ship to the squadron
183     * @param ship the ship to add
184     */
185     Squadron& operator+=(const Ship& ship);
186     /**
187     * Removes a ship from the squadron
188     * @param ship the ship to remove
189     */
190     Squadron& operator-=(const Ship& ship);
191
192     /**
193     * Adds a ship to a copy of the squadron
194     * @param ship the ship to add
195     * @return the copy of the squadron with the ship added
196     */
197     Squadron operator+(Ship& ship) const;
198     /**
199     * Removes a ship from a copy of the squadron
200     * @param ship the ship to remove
201     * @return the copy of the squadron with the ship removed
202     */
203     Squadron operator-(const Ship& ship) const;
204     /**
205     * Get the member of the squadron at the given index
206     *
207     * @param index the index of the member to get
208     * @return Member* the member of the squadron at the given index
209     */
```

```
210     const Ship& operator[](unsigned index) const;
211
212     /**
213     * Get the size of the squadron
214     * @return the size of the squadron
215     */
216     size_t getSize() const;
217
218     /**
219     * Get the maximum speed this squadron can go at
220     *
221     * @return the maximum speed this squadron can go at
222     */
223     unsigned int getMaximumSpeed() const;
224     /**
225     * Get the total weight of this squadron
226     *
227     * @return the total weight of this squadron
228     */
229     double getTotalWeight() const;
230     /**
231     * Get the consumption of the squadron in tons
232     * @param distance distance in mio kilometers
233     * @param speed speed in MGHT
234     * @return the consumption of the squadron in tons
235     * @throw invalid_argument exception
236     */
237     double getConsumption(unsigned distance, unsigned speed) const;
238 };
239
240 #endif //POA_L2_SQUADRON_HPP
241
```

```cpp
#ifndef POA_L2_CARGOSHIP_HPP
#define POA_L2_CARGOSHIP_HPP

#include "Ship.hpp"

/**
 * Represent a spaceship with a cargo
 * @link Ship
 * @version 1.0
 * @author Nelson Jeanrenaud
 * @author André Marques Nora
 */
class CargoShip : public Ship {
    protected:
    double currentCapacity;
    /**
     * @brief Construct a new Cargo Ship object
     *
     * @param serialNumber the serial number of the ship
     * @param currentCapacity the current capacity of the ship
     */
    CargoShip(unsigned int serialNumber, double currentCapacity);
    public:
    /**
     * Get max capacity of the ship in tons
     * @return max capacity of the ship in tons
     */
    virtual double getMaxCapacity() const = 0;
    /**
     * Get the current capacity of the ship in tons
     * @return the current capacity of the ship in tons
     */
    double getCurrentCapacity() const;

    /**
     * Writes the details of the squadron into the output stream.
     * @param out output stream
     * @return output stream
     */
    std::ostream& toStream(std::ostream &out) const;
};


#endif //POA_L2_CARGOSHIP_HPP

```

```cpp
#include "Headers/TieLn.hpp"

unsigned int TieLn::serialNumberCounter = 0;

unsigned int TieLn::getNextSerialNumber() {
    return ++serialNumberCounter;
}

double TieLn::getModelWeight() const {
    return 6;
}

unsigned int TieLn::getModelSpeedMax() const {
    return 100;
}

std::string TieLn::getModel() const {
    return "TIE/LN";
}

TieLn::TieLn() : Ship(getNextSerialNumber()) {}


```

```hpp
#ifndef POA_L2_DESTROYER_HPP
#define POA_L2_DESTROYER_HPP

#include "CargoShip.hpp"

/**
 * Represent a specific model of a Star Wars spaceship cargo
 * @link CargoShip
 * @version 1.0
 * @author Nelson Jeanrenaud
 * @author André Marques Nora
 */
class Destroyer : public CargoShip {
    /// Count of the number of ships created
    static unsigned int serialNumberCounter;
    unsigned int getNextSerialNumber() override;
public:
    /**
     * Construct a new Destroyer ship object
     * @param currentCapacity - current capacity of the ship in tons
     * @throw invalid_argument exception
     */
    explicit Destroyer(double currentCapacity);

    /**
     * Get max capacity of the ship in tons
     * @return max capacity of the ship in tons
     */
    double getMaxCapacity() const override;

    /**
     * Get the weight of the ship in tons
     * @return the weight of the ship in tons
     */
    double getModelWeight() const override;

    /**
     * Get the model of the ship
     * @return the model of the ship
     */
    std::string getModel() const override;

    /**
     * Get the maximum speed of the ship model
     * @return the maximum speed of the ship model
     */
    unsigned int getModelSpeedMax() const override;
};


#endif //POA_L2_DESTROYER_HPP

```

```cpp
#ifndef POA_L2_TIELN_HPP
#define POA_L2_TIELN_HPP


#include "Ship.hpp"
/**
 * Represent a specific model of a Star Wars spaceship
 * @link Ship
 * @version 1.0
 * @author Nelson Jeanrenaud
 * @author André Marques Nora
 */
class TieLn : public Ship {
    /// Count of the number of ships created
    static unsigned int serialNumberCounter;
    /**
     * Get the Next Serial Number object. Increments the counter
     *
     * @return the next serial number
     */
    unsigned int getNextSerialNumber() override;
    public:
        /**
         * Construct a new TieLn object
         */
        TieLn();
        std::string getModel() const override;
        unsigned int getModelSpeedMax() const override;
        double getModelWeight() const override;
};


#endif //POA_L2_TIELN_HPP

```

```cpp
#include "Headers/Ship.hpp"
#include <cmath>
#include <iomanip>

Ship::Ship(unsigned int serialNumber) : nickname(""), serialNumber(serialNumber) {}

void Ship::setNickname(const std::string &newNickname) {
    if (newNickname.empty()) {
        nickname = "";
    } else {
        nickname = newNickname;
    }
}

double Ship::getConsumption(unsigned distance, unsigned speed) const {
   if(speed > this->getModelSpeedMax() || speed <= 0)
      throw std::invalid_argument("Speed must be greater than 0 and less than or equal to max speed");
   return std::cbrt(getModelWeight()) / 2 * log10(getModelWeight() * speed) * log10(distance + 1);
}
std::ostream& Ship::toStream(std::ostream &out) const {
    out << std::fixed << std::setprecision(2) << (nickname.empty() ? "" : nickname + " ");
    out << "[" << getModel() << " #" << serialNumber << "]\n"
    << " weight : " << getModelWeight() << " tons\n"
    << " max speed : " << getModelSpeedMax() << " MGLT\n";
    return out;
}

std::ostream& operator<<(std::ostream& out, const Ship& ship) {
    return ship.toStream(out);
}

std::string Ship::getNickname() const {
   return this->nickname;
}

Ship::~Ship() {

}

```

```cpp
#include "Headers/Shuttle.hpp"

unsigned int Shuttle::serialNumberCounter = 0;

unsigned int Shuttle::getNextSerialNumber() {
    return ++serialNumberCounter;
}

Shuttle::Shuttle(double currentCapacity) : CargoShip(getNextSerialNumber(), currentCapacity) {
    if(currentCapacity < 0 || currentCapacity > this->getMaxCapacity()) {
     throw std::invalid_argument("Capacity must be greater than 0 and less than or equal to max capacity");
    }
}

double Shuttle::getMaxCapacity() const {
    return 80.0;
}

double Shuttle::getModelWeight() const {
    return 360.0;
}

std::string Shuttle::getModel() const {
    return "Lambda-class shuttle";
}

unsigned int Shuttle::getModelSpeedMax() const {
    return 54;
}
```

```cpp
#include "Headers/Squadron.hpp"

#include <utility>

Squadron::Squadron(const std::string &name) : name(name), firstMember(nullptr), leader(nullptr) {
}

Squadron::Squadron(const Squadron &squadron) : Squadron::Squadron(squadron.name) {
    /* Copying the squadron. */
    this->copySquad(squadron);
}

Squadron::~Squadron() {
    /* Deleting the members of the squadron. */
    Member *currentMember = firstMember;
    while (currentMember != nullptr) {
        Member *nextMember = currentMember->getNext();
        delete currentMember;
        currentMember = nextMember;
    }
}

Squadron &Squadron::operator=(const Squadron &squadron) {
   this->emptySquad();

   this->copySquad(squadron);

   return *this;
}

void Squadron::setLeader(const Ship& ship) {
    Member *member = getMember(ship);
    if (member == nullptr) {
        throw std::invalid_argument("Ship is not in the squadron");
    }
    leader = member;
}

bool Squadron::isEmpty() const {
    return firstMember == nullptr;
}

Squadron::Member* Squadron::getMember(const Ship &ship) const{
    if(isEmpty()) {
        return nullptr;
    }
    Member *currentMember = firstMember;
    while (currentMember != nullptr) {
        if (&currentMember->getShip() == &ship) {
            return currentMember;
        }
```

```
 52         currentMember = currentMember->getNext();
 53     }
 54     return nullptr;
 55 }
 56
 57 Squadron::Member* Squadron::getMember(unsigned index) const{
 58     size_t size = this->getSize() - 1;
 59    if(index < 0 || index > size)
 60       throw std::invalid_argument("Index out of bound");
 61
 62    Member* m = this->firstMember;
 63    for (int i = 0; i < size - index; ++i) {
 64       if (m->hasNext())
 65          m = m->getNext();
 66    }
 67    return m;
 68 }
 69
 70 void Squadron::addShipToSelf(const Ship &ship) {
 71     if(isEmpty()) {
 72         firstMember = new Member(&ship);
 73     } else if(getMember(ship) != nullptr){
 74         throw std::invalid_argument("Ship is already in squadron");
 75     }
 76     else {
 77         Member* oldFirstMember = firstMember;
 78         firstMember = new Member(&ship);
 79         firstMember->setNext(oldFirstMember);
 80     }
 81 }
 82
 83 Squadron Squadron::addShip(const Ship& ship) const{
 84     Squadron newSquadron(*this);
 85     newSquadron.addShipToSelf(ship);
 86     return newSquadron;
 87 }
 88
 89 void Squadron::removeShipToSelf(const Ship &ship) {
 90     if(isEmpty()) {
 91         throw std::invalid_argument("Squadron is empty");
 92     }
 93     Member *currentMember = firstMember;
 94     Member *previousMember = nullptr;
 95     while (currentMember != nullptr) {
 96         if (&currentMember->getShip() == &ship) {
 97             if (previousMember == nullptr) {
 98                 firstMember = currentMember->getNext();
 99             } else {
100                 previousMember->setNext(currentMember->getNext());
101             }
102             if(currentMember == leader)
103                 leader = nullptr;
104
```

```
105             delete currentMember;
106             return;
107         }
108         previousMember = currentMember;
109         currentMember = currentMember->getNext();
110     }
111     throw std::invalid_argument("Ship is not in the squadron");
112 }
113
114 Squadron Squadron::removeShip(const Ship &ship) const{
115     Squadron newSquadron(*this);
116     newSquadron.removeShipToSelf(ship);
117     return newSquadron;
118 }
119
120 unsigned int Squadron::getMaximumSpeed() const {
121    if(isEmpty())
122       return 0;
123
124     unsigned int maxSpeed = firstMember->getShip().getModelSpeedMax
    ();
125     Member *currentMember = firstMember->getNext();
126     while (currentMember != nullptr) {
127         if (currentMember->getShip().getModelSpeedMax() < maxSpeed) {
128             maxSpeed = currentMember->getShip().getModelSpeedMax();
129         }
130         currentMember = currentMember->getNext();
131     }
132     return maxSpeed;
133 }
134
135 double Squadron::getTotalWeight() const {
136     if(isEmpty()) {
137         return 0;
138     }
139     double totalWeight = 0;
140     Member *currentMember = firstMember;
141     while (currentMember != nullptr) {
142         totalWeight += currentMember->getShip().getModelWeight();
143         currentMember = currentMember->getNext();
144     }
145     return totalWeight;
146 }
147
148 double Squadron::getConsumption(unsigned distance, unsigned speed)
    const {
149     if(isEmpty()) {
150         return 0;
151     }
152     if(speed > getMaximumSpeed()) {
153         throw std::invalid_argument("Speed is greater than this
    squadron's maximum speed");
154     }
```

```
155     double consumption = 0;
156     Member *currentMember = firstMember;
157     while (currentMember != nullptr) {
158         consumption += currentMember->getShip().getConsumption(
    distance, speed);
159         currentMember = currentMember->getNext();
160     }
161     return consumption;
162 }
163 
164 std::ostream& Squadron::toStream(std::ostream &out) const {
165     out << "Squadron: " << name << " :\n";
166     out << " max speed: " << getMaximumSpeed() << " MGLT\n";
167     out << " total weight: " << getTotalWeight() << " tons\n";
168     out << "-- Leader:\n";
169     if(leader != nullptr) {
170         out << this->getLeader();
171     } else {
172         out << "None\n";
173     }
174     out << "\n-- Members:\n";
175    if(firstMember != nullptr) {
176       Squadron::Member *currentMember = firstMember;
177       while (currentMember != nullptr) {
178          if (currentMember != leader) {
179             out << "\n" << currentMember->getShip() << "\n";
180          }
181          currentMember = currentMember->getNext();
182       }
183    } else {
184       out << "None\n";
185    }
186 
187    return out;
188 }
189 
190 Squadron& Squadron::operator+= (const Ship& ship) {
191     addShipToSelf(ship);
192     return *this;
193 }
194 
195 Squadron& Squadron::operator-= (const Ship &ship) {
196     removeShipToSelf(ship);
197     return *this;
198 }
199 
200 /// Member
201 
202 Squadron::Member::Member(const Ship *ship) : ship(ship), next(nullptr
    ) {}
203 
204 Squadron::Member* Squadron::Member::getNext() const{
205     return next;
```

```
206 }
207
208 const Ship& Squadron::Member::getShip() const{
209     return *ship;
210 }
211
212 void Squadron::Member::setNext(Squadron::Member *next) {
213     this->next = next;
214 }
215
216 bool Squadron::Member::hasNext() const{
217     return next != nullptr;
218 }
219
220 /// Friend
221 std::ostream& operator<<(std::ostream& out, const Squadron& squadron
    ) {
222     return squadron.toStream(out);
223 }
224
225 Squadron Squadron::operator+(Ship &ship) const{
226    return addShip(ship);
227 }
228
229 Squadron Squadron::operator-(const Ship &ship) const{
230    return removeShip(ship);
231 }
232
233 const Ship &Squadron::operator[](unsigned index) const{
234     return getMember(index)->getShip();
235 }
236
237 const Ship *Squadron::getLeader() const {
238    if(this->leader == nullptr)
239       return nullptr;
240
241    return &this->leader->getShip();
242 }
243
244 void Squadron::removeLeader() {
245    this->leader = nullptr;
246 }
247
248 size_t Squadron::getSize() const{
249    if(this->isEmpty())
250       return 0;
251
252    Member* m = this->firstMember;
253    unsigned int counter = 1;
254    while (m->hasNext()){
255       counter++;
256       m = m->getNext();
257    }
```

```
258    return counter;
259 }
260
261 void Squadron::emptySquad() {
262    for (int i = 0; i < this->getSize(); ++i) {
263       this->removeShip(this->getMember(i)->getShip()) ;
264    }
265    this->removeLeader();
266 }
267
268 void Squadron::setName(std::string name) {
269    this->name = std::move(name);
270 }
271
272 std::string Squadron::getName() const{
273    return this->name;
274 }
275
276 void Squadron::copySquad(const Squadron &squadron) {
277    Member *currentMember = squadron.firstMember;
278    while (currentMember != nullptr) {
279       addShipToSelf(currentMember->getShip());
280       currentMember = currentMember->getNext();
281    }
282    if(squadron.leader != nullptr)
283       setLeader(squadron.leader->getShip());
284 }
285
```

```cpp
#include <iostream>
#include "Headers/Squadron.hpp"
#include "Headers/TieLn.hpp"
#include "Headers/Shuttle.hpp"
#include "Headers/TieIn.hpp"
#include "Headers/Destroyer.hpp"

int main(){

   /*
    * Test class Ship
    */

   TieLn blackLeader;
   const TieIn blackOne;
   Destroyer blackTwo(320.6);
   Shuttle blackThree(23.4);
   //Shuttle overWeight(90.4);

   blackLeader.setNickname("Black Star");

   std::cout << blackLeader << std::endl;
   std::cout << blackTwo << std::endl;

   std::cout << "Consumption of blackOne for a distance of 1000 and speed at 50 MGLT : "
      << blackOne.getConsumption(1000,50) << std::endl;

   //std::cout << "Consumption of blackTwo for a distance of 1000 and speed at 1000 MGLT : "
   //          << blackTwo.getConsumption(1000,1000) << std::endl;

   std::cout << "Consumption of blackThree for a distance of 1000 and speed at 50 MGLT : "
             << blackThree.getConsumption(1000,50) << std::endl;
/*
   std::cout << "Consumption of blackThree for a distance of -1000 and speed at 50 MGLT : "
             << blackThree.getConsumption(-1000,50) << std::endl;

   std::cout << "Consumption of blackThree for a distance of 1000 and speed at -50 MGLT : "
             << blackThree.getConsumption(1000,-50) << std::endl;

    std::cout << "Consumption of blackThree for a distance of 1000 and speed at 500 MGLT : "
              << blackThree.getConsumption(1000,500) << std::endl;
*/
   std::cout << std::endl;


   /*
    * Test class Squadron
```

```
48     */
49
50    // constructor, add ship to squad and set a leader
51    Squadron blackSquad("Black Squadron");
52
53    std::cout << blackSquad.getName() << std::endl;
54
55    std::cout << blackSquad << std::endl;
56
57    blackSquad += blackLeader;
58    blackSquad += blackOne;
59    blackSquad + blackTwo;
60    blackSquad += blackThree;
61
62    //blackSquad += blackThree;
63    //blackSquad + blackThree;
64
65    blackSquad.setLeader(blackLeader);
66
67    std::cout << blackSquad << std::endl;
68
69    std::cout << "Consumption of Black Squad for a distance of 1000
   and speed at 30 MGLT : "
70       << blackSquad.getConsumption(1000,30) << std::endl;
71    //std::cout << blackSquad.getConsumption(1000,300) << std::endl;
72
73    std::cout << std::endl;
74
75    //constructor by copy, remove ship from squad and remove leader
76    Squadron blackSquad2(blackSquad);
77
78    std::cout << blackSquad2 << std::endl;
79
80    blackSquad2.removeLeader();
81
82    std::cout << "squad2" << blackSquad2 << std::endl;
83
84    blackSquad2 -= blackLeader;
85    blackSquad2 - blackThree;
86
87    //blackSquad2 -= blackLeader;
88    //blackSquad2 - blackLeader;
89
90    std::cout << blackSquad2 << std::endl;
91
92    TieLn blackLeader2;
93    blackLeader2.setNickname("Black Comet");
94
95    //blackSquad2.setLeader(&blackLeader2);
96
97    blackSquad2 += blackLeader2;
98    blackSquad2.setLeader(blackLeader2);
99
```

```
100    std::cout << blackSquad2 << std::endl;
101
102    //operator= and operator[]
103    std::cout << "operator = & []" << std::endl;
104    std::cout << std::endl;
105
106    Destroyer blueLeader(60);
107    Shuttle blueOne(10);
108
109    Squadron blueSquad("Blue Squadron");
110
111    blueSquad.removeLeader();
112
113    blueSquad += blueLeader;
114
115    //blueSquad.setLeader(&blueOne);
116
117    blueSquad += blueOne;
118
119    blueSquad.setLeader(blueLeader);
120    std::cout << blueSquad << std::endl;
121
122    blueSquad -= blueLeader;
123
124    std::cout << "leader " << (blueSquad.getLeader() == nullptr ? "
    test good" : " test failed") << std::endl;
125
126    blueSquad = blackSquad;
127
128    blueSquad.setName("Blue Monkey");
129
130    std::cout << blueSquad << std::endl;
131
132    std::cout << blueSquad[0] << std::endl;
133
134    //std::cout << blueSquad[10] << std::endl;
135
136     std::cout << "Consumption of blacksquad for a distance of 1000
    and speed at 50 MGLT : "
137               << blackSquad.getConsumption(1000,50) << std::endl;
138
139    /* std::cout << "Consumption of blacksquad for a distance of -1000
     and speed at 50 MGLT : "
140               << blackSquad.getConsumption(-1000,50) << std::endl;
141
142     std::cout << "Consumption of blacksquad for a distance of 1000
    and speed at -50 MGLT : "
143               << blackSquad.getConsumption(1000,-50) << std::endl;
144
145     std::cout << "Consumption of blacksquad for a distance of 1000
    and speed at 500 MGLT : "
146               << blackSquad.getConsumption(1000,500) << std::endl;
147 */
```

```
148     const Squadron constantSquad("constant");
149     const Squadron copyConstant(constantSquad);
150     const Squadron copyConstant2(blueSquad);
151     Squadron copyConstant3(constantSquad);
152 
153     const Squadron constantSquad2 = constantSquad;
154     const Squadron constantSquad3(constantSquad + blackThree);
155     const Squadron constantSquad4(blueSquad + blackTwo);
156 
157 
158     return 0;
159 }
160 
```

File - C:\Heig-vd\POA\POA_L2\src\TieIn.cpp

```cpp
#include "Headers/TieIn.hpp"

unsigned int TieIn::serialNumberCounter = 0;

unsigned int TieIn::getNextSerialNumber() {
    return ++serialNumberCounter;
}

double TieIn::getModelWeight() const {
    return 5;
}

unsigned int TieIn::getModelSpeedMax() const {
    return 110;
}

std::string TieIn::getModel() const {
    return "TIE/IN";
}

TieIn::TieIn() : Ship(getNextSerialNumber()) {}

```