File - main.cpp

```cpp
/**
 * @file Implementation of the main program
 *
 * @author Jallon Sarah
 * @author Surdez Quentin
*/

#include <iostream>
#include "src/ship/TIE/TIE.hpp"
#include "src/ship/TIE/TIEInterceptor.hpp"
#include "src/ship/cargo/StarDreadnought.hpp"
#include "src/ship/cargo/Shuttle.hpp"
#include "src/squadron/Squadron.hpp"
#include <vector>
#include <iostream>

using namespace std;

int main() {

   TIE blackLeader;
   blackLeader.setNickName("Black leader");
   TIE blackTwo;
   Shuttle shuttle(23.4);
   Squadron squad("Black Squadron");
   StarDreadnought star(34.5);
   squad += blackLeader;
   squad += blackTwo;
   squad += shuttle;
   squad += star;
   squad.setChief(blackLeader);
   cout << squad << endl;

}
```

File - Ship.cpp

```cpp

#include "Ship.hpp"
#include "src/utils/ostreamUtils.hpp"

#include <utility>
#include <cmath>

Ship::Ship(size_t _factoryNumber) : factoryNumber(_factoryNumber) {}

void Ship::setNickName(const std::string &_nickName) {
    this->nickName = _nickName;
}

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

std::ostream &Ship::formatToStream(std::ostream &os) const {
    os << (nickName ? nickName.value() + " " : "")
    << "[" <<  getFactoryName() << " #" << factoryNumber << "]""\n";
    ostreamUtils::printWeight(getWeight(), os);
    ostreamUtils::printSpeed(getSpeed(), os);
    return os;
}

long double Ship::fuelConsumption(long double distance, unsigned long long speed) const {
    return std::cbrtl(distance) / 2.0 * log10l(getWeight() * static_cast<long double>(speed)) * log10l(distance + 1.0);
}

size_t Ship::getFactoryNumber() const {
    return factoryNumber;
}
```

File - Ship.hpp

```hpp
/**
 * @file Implementation of the Ship class.
 *
 * @author Jallon Sarah
 * @author Surdez Quentin
 */
#pragma once

#include <ostream>
#include <cstdlib>

class Ship;

/**
 * @brief Implement the operator<< to format the Ship class to an output stream
 * @param os the output stream of interest
 * @param ship the ship to format
 * @return the changed output stream
 */
std::ostream &operator<<(std::ostream &os, const Ship &ship);

/**
 * @brief Abstract class representing a spaceship. It can have a nick name and
 * always has a factory number. It also has a specific weight and a specific speed.
 *
 */
class Ship {
private:
   std::optional<std::string> nickName;
   size_t factoryNumber;

public:

   /**
    * @brief Destructor of the Ship class. Virtual so that it can be overwritten
    * by the derived classes at the starting point of the destruction.
    */
   virtual ~Ship() = default;

   /**
    * @brief Format the ship to an output stream
    * @param os the output stream of interest
    * @return the changed output stream
    */
   virtual std::ostream &formatToStream(std::ostream &os) const;

   /**
    * @brief Get the weight of the ship. Pure virtual function.
    * @return the weight of the ship
```

```
50     */
51    virtual long double getWeight() const = 0;
52
53    /**
54     * @brief Get the speed of the ship. Pure virtual function.
55     * @return the speed of the ship
56     */
57    virtual unsigned long long int getSpeed() const = 0;
58
59    /**
60     * @brief Get the factory name of the ship. Pure virtual function.
61     * @return the factory name of the ship
62     */
63    virtual std::string getFactoryName() const = 0;
64
65    /**
66     * @brief Set the nick name of the ship
67     */
68    void setNickName(const std::string &nickName);
69
70    /**
71     * @brief Get the factory number of the ship
72     * @return the factory number of the ship
73     */
74    size_t getFactoryNumber() const;
75
76    /**
77     * @brief Get the fuel consumption of the ship for the given
78     * distance and the given speed
79     * @param distance the distance to travel
80     * @param speed the speed at which the ship will travel
81     */
82     long double fuelConsumption(long double distance, unsigned long
   long int speed) const;
83
84    /**
85     * @brief Constructor of the Ship class
86     * @param _factoryNumber the factory number of the ship
87     */
88    explicit Ship(size_t _factoryNumber);
89
90 };
91
```

File - TIE.cpp

```cpp

#include "TIE.hpp"

size_t TIE::count = 1;

TIE::TIE() : TIEBaseModel(count++) {

}

long double TIE::getWeight() const {
    return 6.0;
}

unsigned long long int TIE::getSpeed() const {
    return 100;
}

std::string TIE::getFactoryName() const {
    return TIEBaseModel::getFactoryName() + "LN";
}

```

```cpp
/**
 * @file Implementation of the TIE class.
 *
 * @author Jallon Sarah
 * @author Surdez Quentin
 */
#pragma once

#include "src/ship/TIE/TIEBaseModel.hpp"

class TIE : public TIEBaseModel {
private:
   static size_t count;

public:

   /**
    * @inheritDoc
    */
   long double getWeight() const override;

   /**
    * @inheritDoc
    */
   unsigned long long int getSpeed() const override;

   /**
    * @inheritDoc
    */
   std::string getFactoryName() const override;

   /**
    * @brief Constructor of TIE
    */
   TIE();

};
```

```cpp

#include "TIEBaseModel.hpp"

TIEBaseModel::TIEBaseModel(size_t _factoryNumber) : Ship(
_factoryNumber) {}

std::string TIEBaseModel::getFactoryName() const {
    return "TIE/";
}


```

```cpp
/**
 * @file Implementation of the TIEBaseModel class.
 *
 * @author Jallon Sarah
 * @author Surdez Quentin
 */
#pragma once

#include "src/ship/Ship.hpp"


/**
 * @brief A class representing a TIE base model.
 */
class TIEBaseModel : public Ship {
protected:
   /**
    * @brief Constructor for the TIEBaseModel class
    * @param _factoryNumber the factory number of the ship
    */
   explicit TIEBaseModel(size_t _factoryNumber);

public:

   /**
    * @inheritDoc
    */
   std::string getFactoryName() const override;

};
```

File - TIEInterceptor.cpp

```cpp

#include "TIEInterceptor.hpp"

size_t TIEInterceptor::count = 1;

TIEInterceptor::TIEInterceptor() : TIEBaseModel(count++) {

}

unsigned long long int TIEInterceptor::getSpeed() const {
    return 110;
}

long double TIEInterceptor::getWeight() const {
    return 5;
}

std::string TIEInterceptor::getFactoryName() const {
    return TIEBaseModel::getFactoryName() + "IN";
}

```

```cpp
/**
 * @file Implementation of the TIEInterceptor class.
 *
 * @author Jallon Sarah
 * @author Surdez Quentin
 */
#pragma once

#include "src/ship/TIE/TIEBaseModel.hpp"

class TIEInterceptor : public TIEBaseModel {

private:
   static size_t count;

public:

   /**
    * @inheritDoc
    */
   long double getWeight() const override;

   /**
    * @inheritDoc
    */
   unsigned long long int getSpeed() const override;

   /**
    * @inheritDoc
    */
   std::string getFactoryName() const override;

   /**
    * @brief Constructor of TIEInterceptor
    */
   TIEInterceptor();

};
```

File - Shuttle.cpp

```cpp
#include "Shuttle.hpp"

size_t Shuttle::count = 1;


Shuttle::Shuttle(long double currentCargo) : CargoShip(count++,
getMaxCargo(), currentCargo) {
}

long double Shuttle::getWeight() const {
    return 360.0 + getCurrentCargo();
}

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

unsigned long long int Shuttle::getSpeed() const {
    return 54.0;
}

long double Shuttle::getMaxCargo() const {
    return 80.0;
}

```

```cpp
/**
 * @file Implementation of the Shuttle class.
 *
 * @author Jallon Sarah
 * @author Surdez Quentin
 */
#pragma once

#include "src/ship/cargo/CargoShip.hpp"

/**
 * @brief A class representing a shuttle. It is a specialization of the CargoShip class.
 */
class Shuttle : public CargoShip {
private:
   static size_t count;

public:

   /**
    * @inheritDoc
    */
   long double getWeight() const override;

   /**
    * @inheritDoc
    */
   unsigned long long int getSpeed() const override;

   /**
    * @inheritDoc
    */
   std::string getFactoryName() const override;

   /**
    * @inheritDoc
    */
   long double getMaxCargo() const override;

   /**
    * @brief Constructor of Shuttle
    * @param currentCargo the current cargo amount
    */
   explicit Shuttle(long double currentCargo);

};
```

```cpp

#include "CargoShip.hpp"
#include "src/utils/ostreamUtils.hpp"

CargoShip::CargoShip(size_t _factoryNumber, long double _maxCargo,
  long double _currentCargo)
    : Ship(_factoryNumber), currentCargo(_currentCargo) {
    if (currentCargo > _maxCargo) {
       throw std::invalid_argument("Current Cargo cannot be bigger than
   maxCargo");
    }
}


std::ostream &CargoShip::formatToStream(std::ostream &os) const {
    Ship::formatToStream(os);
    ostreamUtils::printCargo(currentCargo, getMaxCargo(), os);
    return os;
}

long double CargoShip::getCurrentCargo() const {
    return currentCargo;
}


```

File - CargoShip.hpp

```hpp
/**
 * @file Implementation of the CargoShip class.
 *
 * @author Jallon Sarah
 * @author Surdez Quentin
 */
#pragma once

#include "src/ship/Ship.hpp"

/**
 * @brief Abstract class representing a specialization of Ship that can carry cargo.
 * they have a cargo capacity and a current cargo amount.
 */
class CargoShip : public Ship {
private:
    long double currentCargo;

protected:

    /**
     * @brief Constructor of CargoShip
     * @throw std::invalid_argument if the current cargo is bigger than the max cargo
     * @param _factoryNumber the factory number of the ship
     * @param _maxCargo the max cargo capacity
     * @param _currentCargo the current cargo amount
     */
    explicit CargoShip(size_t _factoryNumber, long double _maxCargo, long double _currentCargo);

public:

    /**
     * @brief Get the max cargo capacity
     * @return the max cargo capacity
     */
    virtual long double getMaxCargo() const = 0;

    /**
     * @brief Get the current cargo amount.
     * @return the current cargo amount.
     */
    long double getCurrentCargo() const;

    /**
     * @inheritDoc
     */
    std::ostream &formatToStream(std::ostream &os) const override;

};

```

```
51
52
53
```

File - StarDreadnought.cpp

```cpp
#include "StarDreadnought.hpp"

size_t StarDreadnought::count = 1;

StarDreadnought::StarDreadnought(long double _currentCargo) :
CargoShip(count++, getMaxCargo(), _currentCargo) {

}

long double StarDreadnought::getWeight() const {
    return 9e9 + getCurrentCargo();
}

unsigned long long int StarDreadnought::getSpeed() const {
    return 40;
}

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

long double StarDreadnought::getMaxCargo() const {
    return 250e3;
}


```

```hpp
/**
 * @file Implementation of the StarDreadnought class.
 *
 * @author Jallon Sarah
 * @author Surdez Quentin
 */
#pragma once

#include "src/ship/cargo/CargoShip.hpp"

/**
 * @brief A class representing a star dreadnought. It is a specialization of the CargoShip class.
 */
class StarDreadnought : public CargoShip {
private:
    static size_t count;

public:

    /**
     * @inheritDoc
     */
    long double getWeight() const override;

    /**
     * @inheritDoc
     */
    unsigned long long int getSpeed() const override;

    /**
     * @inheritDoc
     */
    std::string getFactoryName() const override;

    /**
     * @inheritDoc
     */
    long double getMaxCargo() const override;

    /**
     * @brief Constructor of StarDreadnought
     * @param _currentCargo the current cargo amount
     */
    explicit StarDreadnought(long double _currentCargo);

};
```

File - test.cpp

```cpp

/**
 * @file Implementation of the different tests we make on the classes we coded
 *
 * @author Jallon Sarah
 * @author Surdez Quentin
*/

#include <gtest/gtest.h>
#include "src/ship/TIE/TIE.hpp"
#include "src/ship/cargo/Shuttle.hpp"
#include "src/ship/cargo/StarDreadnought.hpp"
#include "../squadron/Squadron.hpp"
#include "src/ship/TIE/TIEInterceptor.hpp"
#include "../utils/ostreamUtils.hpp"

TEST(ConstructorSquadron, defaultConstructor) {
   Squadron squadron("Squadron 1");
   ASSERT_EQ(squadron.empty(), true);
}

TEST(ConstrucorSquadron, copyConstructor) {
   Squadron squadron("Squadron 1");
   TIEInterceptor interceptor;
   squadron.addShip(interceptor);
   TIEInterceptor interceptor1;
   squadron.addShip(interceptor1);
   Squadron squadron2(squadron);
   std::cout << squadron << std::endl;
   std::cout << squadron2 << std::endl;
   ASSERT_EQ(squadron.empty(), false);
   ASSERT_EQ(squadron2.empty(), false);
}

TEST(ConstructorSquadron, moveConstructor) {
   Squadron squadron("Squadron 1");
   TIEInterceptor interceptor;
   squadron.addShip(interceptor);
   TIEInterceptor interceptor1;
   squadron.addShip(interceptor1);
   Squadron squadron2(std::move(squadron));
   std::cout << squadron << std::endl;
   std::cout << squadron2 << std::endl;
   ASSERT_EQ(squadron.empty(), true);
   ASSERT_EQ(squadron2.empty(), false);
}

TEST(Assignments, CopyAssignment) {
   Squadron squadron("Squadron 1");
   TIEInterceptor interceptor;
   squadron.addShip(interceptor);
   TIEInterceptor interceptor1;
```

```
 53    squadron.addShip(interceptor1);
 54    Squadron squadron2 = squadron;
 55    std::cout << squadron << std::endl;
 56    std::cout << squadron2 << std::endl;
 57    ASSERT_EQ(squadron.empty(), false);
 58    ASSERT_EQ(squadron2.empty(), false);
 59 }
 60
 61 TEST(Assignments, MoveAssignment) {
 62    Squadron squadron("Squadron 1");
 63    TIEInterceptor interceptor;
 64    squadron.addShip(interceptor);
 65    TIEInterceptor interceptor1;
 66    squadron.addShip(interceptor1);
 67    Squadron squadron2 = std::move(squadron);
 68    std::cout << squadron << std::endl;
 69    std::cout << squadron2 << std::endl;
 70    ASSERT_EQ(squadron.empty(), true);
 71    ASSERT_EQ(squadron2.empty(), false);
 72 }
 73
 74 TEST(Operations, AddShipToSelf) {
 75    Squadron squadron("Squadron 1");
 76    TIEInterceptor interceptor;
 77    squadron.addShip(interceptor);
 78    TIEInterceptor interceptor1;
 79    squadron.addShip(interceptor1);
 80    std::cout << squadron << std::endl;
 81    ASSERT_EQ(squadron.empty(), false);
 82 }
 83
 84 TEST(Operations, AddShipCopy) {
 85    Squadron squadron("Squadron 1");
 86    TIEInterceptor interceptor;
 87    Squadron squadron1 = squadron.addShipCopy(interceptor);
 88    std::cout << squadron1 << std::endl;
 89    ASSERT_EQ(squadron.empty(), true);
 90    ASSERT_EQ(squadron1.empty(), false);
 91 }
 92
 93 TEST(Operations, DeleteShip) {
 94    Squadron squadron("Squadron 1");
 95    TIEInterceptor interceptor;
 96    squadron.addShip(interceptor);
 97    std::cout << squadron << std::endl;
 98    squadron.deleteShip(interceptor);
 99    std::cout << squadron << std::endl;
100    ASSERT_EQ(squadron.empty(), true);
101 }
102
103 TEST(Operations, DeleteShipCopy) {
104    Squadron squadron("Squadron 1");
105    TIEInterceptor interceptor;
```

```
106     squadron.addShip(interceptor);
107     std::cout << squadron << std::endl;
108     Squadron squadron1 = squadron.deleteShipCopy(interceptor);
109     std::cout << squadron1 << std::endl;
110     ASSERT_EQ(squadron.empty(), false);
111     ASSERT_EQ(squadron1.empty(), true);
112 }
113
114 TEST(Operations, plusEqual) {
115     Squadron squadron("Squadron 1");
116     TIEInterceptor interceptor;
117     squadron += interceptor;
118     std::cout << squadron << std::endl;
119     ASSERT_EQ(squadron.empty(), false);
120 }
121
122 TEST(Operations, minusEqual) {
123     Squadron squadron("Squadron 1");
124     TIEInterceptor interceptor;
125     squadron += interceptor;
126     std::cout << squadron << std::endl;
127     squadron -= interceptor;
128     std::cout << squadron << std::endl;
129     ASSERT_EQ(squadron.empty(), true);
130 }
131
132 TEST(Operations, plus) {
133     Squadron squadron("Squadron 1");
134     TIEInterceptor interceptor;
135     Squadron squadron1 = squadron + interceptor;
136     std::cout << squadron1 << std::endl;
137     ASSERT_EQ(squadron.empty(), true);
138     ASSERT_EQ(squadron1.empty(), false);
139 }
140
141 TEST(Operations, minus) {
142     Squadron squadron("Squadron 1");
143     TIEInterceptor interceptor;
144     squadron += interceptor;
145     std::cout << squadron << std::endl;
146     Squadron squadron1 = squadron - interceptor;
147     std::cout << squadron1 << std::endl;
148     ASSERT_EQ(squadron.empty(), false);
149     ASSERT_EQ(squadron1.empty(), true);
150 }
151
152 TEST(Getter, getShip) {
153     Squadron squadron("Squadron 1");
154     TIEInterceptor interceptor;
155     squadron += interceptor;
156     std::cout << squadron.get(0) << std::endl;
157 }
158
```

File - test.cpp

```
159 TEST(Getter, getShipOutOfBounds) {
160    Squadron squadron("Squadron 1");
161    TIEInterceptor interceptor;
162    squadron += interceptor;
163    EXPECT_THROW(
164              {
165                try {
166                    std::cout << squadron.get(1) << std::endl;
167                } catch (std::out_of_range &re) {
168                    EXPECT_STREQ("Index out of range", re.what());
169                    throw;
170                }
171              }, std::out_of_range);
172 }
173
174 TEST(Consumption, TotalConsumption) {
175    Squadron squadron("Squadron 1");
176    TIEInterceptor interceptor;
177    squadron += interceptor;
178    Shuttle shuttle(23.4);
179    squadron += shuttle;
180    StarDreadnought star(34.5);
181    squadron += star;
182    std::cout << squadron << std::endl;
183    ASSERT_FLOAT_EQ(squadron.fuelConsumption(squadron.getWeight(),
    squadron.getMaximumSpeed()), 186796.56);
184 }
185
186 TEST(Consumption, TotalConsumptionIfEmpty) {
187    Squadron squadron("Squadron 1");
188    EXPECT_THROW(
189              {
190                try {
191                    std::cout << squadron.fuelConsumption(squadron.
    getWeight(), squadron.getMaximumSpeed()) << std::endl;
192                } catch (std::runtime_error &re) {
193                    EXPECT_STREQ("Squadron is empty", re.what());
194                    throw;
195                }
196              }, std::runtime_error);
197 }
198
199 TEST(Chief, setChief) {
200    Squadron squadron("Squadron 1");
201    TIEInterceptor interceptor;
202    squadron += interceptor;
203    squadron.setChief(interceptor);
204    std::cout << squadron << std::endl;
205 }
206
207 TEST(Chief, setChiefIfAlreadyChief) {
208    Squadron squadron("Squadron 1");
209    TIEInterceptor interceptor;
```

```
210    Shuttle shuttle(23.4);
211    squadron += interceptor;
212    squadron.setChief(interceptor);
213    std::cout << squadron << std::endl;
214    squadron.setChief(shuttle);
215    std::cout << squadron << std::endl;
216 }
217
218 TEST(Chief, removeChief) {
219    Squadron squadron("Squadron 1");
220    TIEInterceptor interceptor;
221    squadron += interceptor;
222    squadron.setChief(interceptor);
223    std::cout << squadron << std::endl;
224    squadron.removeChief();
225    std::cout << squadron << std::endl;
226 }
227
228 TEST(Chief, removeChiefTwice) {
229    Squadron squadron("Squadron 1");
230    TIEInterceptor interceptor;
231    squadron += interceptor;
232    squadron.setChief(interceptor);
233    std::cout << squadron << std::endl;
234    squadron.removeChief();
235    std::cout << squadron << std::endl;
236    squadron.removeChief();
237    std::cout << squadron << std::endl;
238 }
239
240 TEST(Others, maximumSpeed) {
241    Squadron squadron("Squadron 1");
242    TIEInterceptor interceptor;
243    squadron += interceptor;
244    Shuttle shuttle(23.4);
245    squadron += shuttle;
246    StarDreadnought star(34.5);
247    squadron += star;
248    std::cout << squadron << std::endl;
249    ASSERT_FLOAT_EQ(squadron.getMaximumSpeed(), 40.0);
250 }
251
252 TEST(Others, totalWeight) {
253    Squadron squadron("Squadron 1");
254    TIEInterceptor interceptor;
255    squadron += interceptor;
256    Shuttle shuttle(23.4);
257    squadron += shuttle;
258    std::cout << squadron << std::endl;
259    ASSERT_FLOAT_EQ(squadron.getWeight(), 388.39999);
260 }
261
262 TEST(Others, empty) {
```

```
263     Squadron squadron("Squadron 1");
264     ASSERT_EQ(squadron.empty(), true);
265     TIE tie;
266     squadron += tie;
267     ASSERT_EQ(squadron.empty(), false);
268 }
269
270 int main(int argc, char **argv) {
271 testing::InitGoogleTest(&argc, argv);
272 return RUN_ALL_TESTS();
273 }
274
275
```

File - ostreamUtils.cpp

```cpp

#include <iomanip>
#include "ostreamUtils.hpp"

std::ostream &ostreamUtils::printWeight(long double weight, std::ostream &os) {
    fixed(os);
    os << std::setprecision(2) << "weight: " << weight << " tons\n";
    return os;
}

std::ostream &ostreamUtils::printSpeed(unsigned long long speed, std::ostream &os) {
    os << "max speed: " << speed << " MGLT\n";
    return os;
}

std::ostream &ostreamUtils::printCargo(long double currentCargo, long double maxCargo, std::ostream &os) {
    fixed(os);
    os << std::setprecision(1) << "cargo: " << currentCargo << " tons ( max : " << maxCargo << ")\n";
    return os;
}

std::ostream &ostreamUtils::fixed(std::ostream &os) {
    os << std::fixed;
    return os;
}
```

File - ostreamUtils.hpp

```hpp
/**
 * @file Implementation of the namespace ostreamUtils
 *
 * @author Jallon Sarah
 * @author Surdez Quentin
*/
#pragma once

#include <ostream>

/**
 * @brief A namespace containing functions to print information
 */
namespace ostreamUtils {

   /**
    * @brief Print the weight with specific precision of 2
    * @param weight the weight to print
    * @param os the output stream
    * @return the changed output stream
    */
   std::ostream &printWeight(long double weight, std::ostream &os);

   /**
    * @brief Print the speed with specific precision of 0
    * @param speed the speed to print
    * @param os the output stream
    * @return the changed output stream
    */
   std::ostream &printSpeed(unsigned long long speed, std::ostream &os
);

   /**
    * @brief Print the current cargo and the max cargo with specific
precision of 1
    * @param currentCargo the current cargo to print
    * @param maxCargo the max cargo to print
    * @param os the output stream
    * @return the changed output stream
    */
   std::ostream &printCargo(long double currentCargo, long double
maxCargo, std::ostream &os);

   /**
    * @brief Set the output stream to fixed
    * @param os the output stream
    * @return the changed output stream
    */
   std::ostream &fixed(std::ostream &os);

}


```

File - Squadron.cpp

```cpp

#include "Squadron.hpp"
#include "src/utils/ostreamUtils.hpp"

Squadron operator+(const Squadron &rhs, Ship &lhs) {
    return Squadron(rhs).addShip(lhs);
}

Squadron operator-(const Squadron &rhs, Ship &lhs) {
    return Squadron(rhs).deleteShip(lhs);
}

std::ostream &operator<<(std::ostream &os, const Squadron &squadron) {
    return squadron.formatToStream(os);
}

Squadron::Squadron(const std::string &_name) {
    initSquadron(_name);
}

Squadron::~Squadron() {
    deleteSquadron();
}

Squadron::Squadron(const Squadron &other) {
    initSquadron(other.name, other.chief);
    copySquadron(other);
}

Squadron::Squadron(Squadron &&other) noexcept {
    initSquadron(other.name);
    exchange(other);
}

Squadron &Squadron::operator=(const Squadron &other) {
    if (this != &other) {
        deleteSquadron();
        initSquadron(other.name, other.chief);
        copySquadron(other);
    }
    return *this;
}

Squadron &Squadron::operator=(Squadron &&other) noexcept {
    if (this != &other) {
        deleteSquadron();
        initSquadron(other.name);
        exchange(other);
    }
    return *this;
}

Squadron &Squadron::addShip(reference ship) {
```

```
 54    if (head != nullptr) {
 55       Node *current = head;
 56       while (current->next != nullptr) {
 57          if (current->next->value == &ship || current->value == &ship
    ) {
 58             return *this;
 59          }
 60          current = current->next;
 61       }
 62       current->next = new Node{&ship, nullptr};
 63    } else {
 64       head = new Node{&ship, nullptr};
 65    }
 66    size++;
 67    return *this;
 68 }
 69
 70 Squadron Squadron::addShipCopy(reference ship) const {
 71    return Squadron(*this).addShip(ship);
 72 }
 73
 74 Squadron &Squadron::deleteShip(reference ship) {
 75    Node *current = head;
 76    while (current != nullptr) {
 77       if (current->value == &ship) {
 78          Node *toDelete = current;
 79          if (current->next != nullptr) {
 80             current->next = current->next->next;
 81          }
 82          if (toDelete->value == chief) {
 83             removeChief();
 84          }
 85          delete toDelete;
 86          if (toDelete == head) {
 87             head = nullptr;
 88          }
 89          size--;
 90          break;
 91       }
 92       current = current->next;
 93    }
 94    return *this;
 95 }
 96
 97 Squadron Squadron::deleteShipCopy(reference ship) const {
 98    return Squadron(*this).deleteShip(ship);
 99 }
100
101 Squadron &Squadron::operator+=(reference lhs) {
102    return addShip(lhs);
103 }
104
105 Squadron &Squadron::operator-=(reference lhs) {
```

```
106     return deleteShip(lhs);
107 }
108 
109 void Squadron::setChief(reference newChief) {
110     if (chief != nullptr || chief == &newChief) {
111         return;
112     }
113     if (!contains(newChief)) {
114         addShip(newChief);
115     }
116     chief = &newChief;
117 }
118 
119 void Squadron::removeChief() {
120     chief = nullptr;
121 }
122 
123 Squadron::reference Squadron::get(size_t i) const {
124     if (i >= size) {
125         throw std::out_of_range("Index out of range");
126     }
127     Node *current = head;
128     for (size_t j = 0; j < i; ++j) {
129         current = current->next;
130     }
131     return *current->value;
132 }
133 
134 void Squadron::setName(const std::string &newName) {
135     name = newName;
136 }
137 
138 bool Squadron::empty() const {
139     return size == 0;
140 }
141 
142 long double Squadron::fuelConsumption(long double distance, unsigned
    long long int speed) const {
143 
144 
145     if (distance < 0 || speed <= 0) {
146         throw std::invalid_argument("Distance and speed must be
    positive");
147     }
148 
149     if (speed > getMaximumSpeed()) {
150         throw std::invalid_argument("Speed cannot be higher than the
    maximum speed at which the squadron travel");
151     }
152 
153     if (empty()) {
154         throw std::runtime_error("Squadron is empty");
155     }
```

```
156
157     long double totalFuel = 0;
158     Node *current = head;
159     for (size_t i = 0; i < size; i++) {
160         totalFuel += current->value->fuelConsumption(distance, speed);
161         current = current->next;
162     }
163     return totalFuel;
164 }
165
166 unsigned long long int Squadron::getMaximumSpeed() const {
167     unsigned long long minSpeed = std::numeric_limits<unsigned long
    long>::max();
168     Node *current = head;
169     while (current != nullptr) {
170         minSpeed = std::min(minSpeed, current->value->getSpeed());
171         current = current->next;
172     }
173     return minSpeed;
174 }
175
176 long double Squadron::getWeight() const {
177     long double totalWeight = 0.0;
178     Node *current = head;
179     while (current != nullptr) {
180         totalWeight += current->value->getWeight();
181         current = current->next;
182     }
183     return totalWeight;
184 }
185
186 std::ostream &Squadron::formatToStream(std::ostream &os) const {
187     if (!empty()) {
188         os << "Squadron " << name << "\n";
189         ostreamUtils::printSpeed(getMaximumSpeed(), os);
190         ostreamUtils::printWeight(getWeight(), os);
191         if (chief != nullptr) {
192             os << "--Leader:\n" << *chief;
193         }
194         if (head->value == chief && head->next != nullptr || head->
    value != chief) {
195             os << "--Members:\n";
196             Node *current = head;
197             while (current != nullptr) {
198                 if (current->value == chief) {
199                     current = current->next;
200                     continue;
201                 }
202                 os << *current->value;
203                 current = current->next;
204             }
205         }
206     } else {
```

```
207         os << "Squadron is empty\n";
208     }
209     return os;
210 }
211 
212 // Private methods
213 
214 void Squadron::initSquadron(const std::string &n, pointer _chief) {
215     chief = _chief;
216     name = n;
217 }
218 
219 
220 void Squadron::deleteSquadron() {
221     Node *current = head;
222     while (current != nullptr) {
223         Node *next = current->next;
224         delete current;
225         current = next;
226     }
227 }
228 
229 void Squadron::copySquadron(const Squadron &other) {
230     if (head == other.head) {
231         return;
232     }
233     for (size_t i = 0; i < other.size; i++) {
234         addShip(other.get(i));
235     }
236 }
237 
238 void Squadron::exchange(Squadron &other) {
239     head = std::exchange(other.head, nullptr);
240     name = std::exchange(other.name, std::string{});
241     chief = std::exchange(other.chief, nullptr);
242     size = std::exchange(other.size, 0);
243 }
244 
245 bool Squadron::contains(Squadron::reference ship) {
246     Node *current = head;
247     while (current != nullptr) {
248         if (current->value == &ship) {
249             return true;
250         }
251         current = current->next;
252     }
253     return false;
254 }
255 
```

File - Squadron.hpp

```hpp
/**
 * @file Implementation of the Squadron class.
 *
 * @author Jallon Sarah
 * @author Surdez Quentin
 */
#pragma once

#include <string>
#include "src/ship/Ship.hpp"

class Squadron;

/**
 * @brief Implement the operator+ to add a ship to a squadron
 * @param rhs the right hand side of the operator which is a squadron
 * @param lhs the left hand side of the operator which is a ship
 * @return a new squadron with the ship added
 */
[[nodiscard]] Squadron operator+(const Squadron &rhs, Ship &lhs);

/**
 * @brief Implement the operator- to remove a ship from a squadron
 * @param rhs the right hand side of the operator which is a squadron
 * @param lhs the left hand side of the operator which is a shipnd
 * @return a new squadron with the ship removed
 */
[[nodiscard]] Squadron operator-(const Squadron &rhs, Ship &lhs);

/**
 * @brief Implement the operator<< to format the squadron to an output stream
 * @param os the output stream of interest
 * @param squadron the squadron to format
 * @return the output stream changed
 */
std::ostream &operator<<(std::ostream &os, const Squadron &squadron);

/**
 * @brief Squadron class is used to represent a squadron of ships. It is a linked list of ships.
 * Each squadron has a name, a potential chief and a list of ships that are a part of the
 * squadron. The rule of five has been followed to ensure that the class corresponds to
 * the extended canonical form.
 */
class Squadron {
public:
    using value_type = Ship;
    using pointer = Ship *;
    using reference = Ship &;
    using const_reference = const Ship &;
```

```
50
51      /**
52       * @brief Constructor of the Squadron class
53       * @param _name the name of the squadron
54       */
55      explicit Squadron(const std::string &_name);
56
57      // Rule of Five
58
59      /**
60       * Destructor of the Squadron class
61       */
62      ~Squadron();
63
64      /**
65       * @brief Copy constructor of the Squadron class
66       * @param other the squadron to copy
67       */
68      Squadron(const Squadron &other);
69
70      /**
71       * @brief Move constructor of the Squadron class
72       * @param other the squadron to move
73       */
74      Squadron(Squadron &&other) noexcept;
75
76      /**
77       * @brief Copy assignment operator of the Squadron class
78       * @param other the squadron to copy
79       * @return the instance of the squadron which has the elements
80       * of the copied squadron
81       */
82      Squadron &operator=(const Squadron &other);
83
84      /**
85       * @brief Move assignment operator of the Squadron class
86       * @param other the squadron to move
87       * @return the instance of the squadron which has the elements
88       * of the moved squadron
89       */
90      Squadron &operator=(Squadron &&other) noexcept;
91
92      /**
93       * @brief Add a ship to the squadron
94       * @param ship ship to add to the squadron
95       * @return the instance of the squadron with the ship added
96       */
97      Squadron &addShip(reference ship);
98
99      /**
100      * @brief Add a ship to a copied instance of the squadron
101      * @param ship ship to add to the new squadron
102      * @return the newly created instance of the squadron
```

```
103     */
104    [[nodiscard]] Squadron addShipCopy(reference ship) const;
105
106    /**
107     * @brief Delete a ship from the squadron
108     * @param ship ship to delete
109     * @return the instance of the squadron with the ship deleted
110     */
111    Squadron &deleteShip(reference ship);
112
113    /**
114     * @brief Delete a ship from a copied instance of the squadron
115     * @param ship ship to delete from the new squadron
116     * @return the newly created instance of the squadron
117     */
118    [[nodiscard]] Squadron deleteShipCopy(reference ship) const;
119
120    /**
121     * @brief Implement the operator+= on the squadron to add a ship
122     * @param lhs the left hand side of the operator which is a ship
123     * @return the instance of the squadron with the ship added
124     */
125    Squadron &operator+=(reference lhs);
126
127    /**
128     * @brief Implement the operator-= on the squadron to remove a
    ship
129     * @param lhs the left hand side of the operator which is a ship
130     * @return the instance of the squadron with the ship removed
131     */
132    Squadron &operator-=(reference lhs);
133
134    /**
135     * @brief Sets chief to the squadron. If the chief is not a part
136     * of the squadron, it is added to the squadron.
137     * @param newChief the ship to set as the chief
138     */
139    void setChief(reference newChief);
140
141    /**
142     * @brief Resets the chief of the squadron. Does not remove the
    ship from the squadron.
143     */
144    void removeChief();
145
146    /**
147     * @brief Get the i-th ship of the squadron
148     * @throw std::out_of_range if i is out of the index
149     * @param i index of the ship of interest
150     * @return a reference on the ship of interest
151     */
152    [[nodiscard]] reference get(size_t i) const;
153
```

```
154     /**
155      * @brief Change the name of the squadron
156      * @param newName the new name of the squadron
157      */
158     void setName(const std::string &newName);
159 
160     /**
161      * @brief Checks if the squadron is empty
162      * @return true if it is, false if it is not
163      */
164     bool empty() const;
165 
166     /**
167      * @brief Method to determine the fuel consumption of the squadron
168      * @throw std::run_time_error if Squadron is empty
169      * @throw std::invalid_argument if distance or speed is negative
    or if speed is greater than the maximum speed
170      * @param distance the distance the squadron will travel
171      * @param speed the speed at which the squadron will travel
172      * @return the estimate consumption for the whole squadron
173      */
174     long double fuelConsumption(long double distance, unsigned long
    long int speed) const;
175 
176     /**
177      * @brief Get the maximum speed at which the squadron can travel
178      * @return the maximum speed at which the squadron can travel
179      */
180     unsigned long long int getMaximumSpeed() const;
181 
182     /**
183      * @brief Get the total weight of the squadron
184      * @return the total weight of the squadron
185      */
186     long double getWeight() const;
187 
188     /**
189      * @brief Format the squadron for an output stream
190      * @param os the output stream of interest
191      * @return the changed output stream
192      */
193     std::ostream &formatToStream(std::ostream &os) const;
194 
195 private:
196 
197     struct Node {
198        pointer value;
199        Node *next;
200     };
201 
202     Node *head{};
203     std::string name;
204     pointer chief{};
```

```
205     size_t size{};
206
207
208
209     void initSquadron(const std::string &n, pointer _chief = nullptr);
210
211     /**
212      * @brief Helper methode to delete the squadron
213      */
214     void deleteSquadron();
215
216     /**
217      * @brief Helper method to copy a squadron
218      * @param other the squadron to copy
219      */
220     void copySquadron(const Squadron &other);
221
222     /**
223      * @brief Helper method to exchange every attribute from
224      * one squadron to another
225      * @param other the squadron to exchange from
226      */
227     void exchange(Squadron &other);
228
229     /**
230      * @brief Helper method to check if the squadron contains a ship
231      * @param ship the ship of interest
232      * @return true if it is in the squadron, false if it is not
233      */
234     bool contains(reference ship);
235
236 };
237
238
239
```