

```
1 /**
2  * @file Implementation of the main program
3  *
4  * @author Jallon Sarah
5  * @author Surdez Quentin
6 */
7
8 #include <iostream>
9 #include "src/ship/TIE/TIE.hpp"
10 #include "src/ship/TIE/TIEInterceptor.hpp"
11 #include "src/ship/cargo/StarDreadnought.hpp"
12 #include "src/ship/cargo/Shuttle.hpp"
13 #include "src/squadron/Squadron.hpp"
14 #include <vector>
15 #include <iostream>
16
17 using namespace std;
18
19 int main() {
20
21     TIE blackLeader;
22     blackLeader.setNickName("Black Leader");
23     TIE blackTwo;
24     Shuttle shuttle(23.4);
25     Squadron squad("Black Squadron");
26     StarDreadnought star(34.5);
27     squad += blackLeader;
28     squad += blackTwo;
29     squad += shuttle;
30     squad += star;
31     squad.setChief(blackLeader);
32     cout << squad << endl;
33
34 }
```

```
1
2 #include "Ship.hpp"
3 #include "src/utils/ostreamUtils.hpp"
4
5 #include <utility>
6 #include <cmath>
7
8 Ship::Ship(size_t _factoryNumber) : factoryNumber(_factoryNumber) {}
9
10 void Ship::setNickName(const std::string &_nickName) {
11     this->nickName = _nickName;
12 }
13
14 std::ostream &operator<<(std::ostream &os, const Ship &ship) {
15     return ship.formatToStream(os);
16 }
17
18 std::ostream &Ship::formatToStream(std::ostream &os) const {
19     os << (nickName ? nickName.value() + " " : "") 
20     << "[" << getFactoryName() << " #" << factoryNumber << "]""\n";
21     ostreamUtils::printWeight(getWeight(), os);
22     ostreamUtils::printSpeed(getSpeed(), os);
23     return os;
24 }
25
26 size_t Ship::getFactoryNumber() const {
27     return factoryNumber;
28 }
```

```
1  /**
2   * @file Implementation of the Ship class.
3   *
4   * @author Jallon Sarah
5   * @author Surdez Quentin
6   */
7 #pragma once
8
9 #include <iostream>
10 #include <cstdlib>
11
12 class Ship;
13
14 /**
15  * @brief Implement the operator<< to format the Ship class to an
16  * output stream
17  * @param os the output stream of interest
18  * @param ship the ship to format
19  * @return the changed output stream
20 */
21 std::ostream &operator<<(std::ostream &os, const Ship &ship);
22
23 /**
24  * @brief Abstract class representing a spaceship. It can have a nick
25  * name and
26  * always has a factory number. It also has a specific weight and a
27  * specific speed.
28 */
29
30 class Ship {
31 private:
32     std::optional<std::string> nickname;
33     size_t factoryNumber;
34
35 public:
36     /**
37      * @brief Destructor of the Ship class. Virtual so that it can be
38      * overwritten
39      * by the derived classes at the starting point of the destruction.
40      */
41     virtual ~Ship() = default;
42
43     /**
44      * @brief Format the ship to an output stream
45      * @param os the output stream of interest
46      * @return the changed output stream
47      */
48     virtual std::ostream &formatToStream(std::ostream &os) const;
49
50     /**
51      * @brief Get the weight of the ship. Pure virtual function.
52      * @return the weight of the ship
53  }
```

```
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 Constructor of the Ship class
78      * @param _factoryNumber the factory number of the ship
79      */
80     explicit Ship(size_t _factoryNumber);
81
82 };
83
```

```
1
2 #include "TIE.hpp"
3
4 size_t TIE::count = 1;
5
6 TIE::TIE() : TIEBaseModel(count++) {
7
8 }
9
10 long double TIE::getWeight() const {
11     return 6.0;
12 }
13
14 unsigned long long int TIE::getSpeed() const {
15     return 100;
16 }
17
18 std::string TIE::getFactoryName() const {
19     return TIEBaseModel::getFactoryName() + "LN";
20 }
21
```

```
1 /**
2  * @file Implementation of the TIE class.
3  *
4  * @author Jallon Sarah
5  * @author Surdez Quentin
6  */
7 #pragma once
8
9 #include "src/ship/TIE/TIEBaseModel.hpp"
10
11 class TIE : public TIEBaseModel {
12 private:
13     static size_t count;
14
15 public:
16
17     /**
18      * @inheritDoc
19      */
20     long double getWeight() const override;
21
22     /**
23      * @inheritDoc
24      */
25     unsigned long long int getSpeed() const override;
26
27     /**
28      * @inheritDoc
29      */
30     std::string getFactoryName() const override;
31
32     /**
33      * @brief Constructor of TIE
34      */
35     TIE();
36
37 };
38
39
40
```

```
1  
2 #include "TIEBaseModel.hpp"  
3  
4 TIEBaseModel::TIEBaseModel(size_t _factoryNumber) : Ship(  
    _factoryNumber) {}  
5  
6 std::string TIEBaseModel::getFactoryName() const {  
7     return "TIE/";  
8 }  
9  
10
```

```
1 /**
2  * @file Implementation of the TIEBaseModel class.
3  *
4  * @author Jallon Sarah
5  * @author Surdez Quentin
6  */
7 #pragma once
8
9 #include "src/ship/Ship.hpp"
10
11
12 /**
13  * @brief A class representing a TIE base model.
14  */
15 class TIEBaseModel : public Ship {
16 protected:
17     /**
18     * @brief Constructor for the TIEBaseModel class
19     * @param _factoryNumber the factory number of the ship
20     */
21     explicit TIEBaseModel(size_t _factoryNumber);
22
23 public:
24     /**
25     * @inheritDoc
26     */
27     std::string getFactoryName() const override;
28
29 };
30
31
32
33
34
```

```
1
2 #include "TIEInterceptor.hpp"
3
4 size_t TIEInterceptor::count = 1;
5
6 TIEInterceptor::TIEInterceptor() : TIEBaseModel(count++) {
7
8 }
9
10 unsigned long long int TIEInterceptor::getSpeed() const {
11     return 110;
12 }
13
14 long double TIEInterceptor::getWeight() const {
15     return 5;
16 }
17
18 std::string TIEInterceptor::getFactoryName() const {
19     return TIEBaseModel::getFactoryName() + "IN";
20 }
21
```

```
1 /**
2  * @file Implementation of the TIEInterceptor class.
3  *
4  * @author Jallon Sarah
5  * @author Surdez Quentin
6  */
7 #pragma once
8
9 #include "src/ship/TIE/TIEBaseModel.hpp"
10
11 class TIEInterceptor : public TIEBaseModel {
12
13 private:
14     static size_t count;
15
16 public:
17
18     /**
19      * @inheritDoc
20      */
21     long double getWeight() const override;
22
23     /**
24      * @inheritDoc
25      */
26     unsigned long long int getSpeed() const override;
27
28     /**
29      * @inheritDoc
30      */
31     std::string getFactoryName() const override;
32
33     /**
34      * @brief Constructor of TIEInterceptor
35      */
36     TIEInterceptor();
37
38 };
39
40
41
42
```

```
1 #include "Shuttle.hpp"
2
3 size_t Shuttle::count = 1;
4
5
6 Shuttle::Shuttle(long double currentCargo) : CargoShip(count++,
    getMaxCargo(), currentCargo) {
7 }
8
9 long double Shuttle::getWeight() const {
10     return 360.0 + getCurrentCargo();
11 }
12
13 std::string Shuttle::getFactoryName() const {
14     return "Lambda-class shuttle";
15 }
16
17 unsigned long long int Shuttle::getSpeed() const {
18     return 54.0;
19 }
20
21 long double Shuttle::getMaxCargo() const {
22     return 80.0;
23 }
24
```

```
1 /**
2  * @file Implementation of the Shuttle class.
3  *
4  * @author Jallon Sarah
5  * @author Surdez Quentin
6  */
7 #pragma once
8
9 #include "src/ship/cargo/CargoShip.hpp"
10
11 /**
12  * @brief A class representing a shuttle. It is a specialization of
13  * the CargoShip class.
14  */
15 class Shuttle : public CargoShip {
16 private:
17     static size_t count;
18 protected:
19     /**
20      * @brief Constructor of Shuttle
21      * @param currentCargo the current cargo amount
22      */
23     explicit Shuttle(long double currentCargo);
24
25 public:
26     /**
27      * @inheritDoc
28      */
29     long double getWeight() const override;
30
31     /**
32      * @inheritDoc
33      */
34     unsigned long long int getSpeed() const override;
35
36     /**
37      * @inheritDoc
38      */
39     std::string getFactoryName() const override;
40
41     /**
42      * @inheritDoc
43      */
44     long double getMaxCargo() const override;
45
46 }
47
48 };
49
50
51
```

```
1
2 #include "CargoShip.hpp"
3 #include "src/utils/ostreamUtils.hpp"
4
5 CargoShip::CargoShip(size_t _factoryNumber, long double _maxCargo,
6                      long double _currentCargo)
7     : Ship(_factoryNumber), currentCargo(_currentCargo) {
8     if (currentCargo > _maxCargo) {
9         throw std::invalid_argument("Current Cargo cannot be bigger than
10            maxCargo");
11    }
12 }
13 std::ostream &CargoShip::formatToStream(std::ostream &os) const {
14     Ship::formatToStream(os);
15     ostreamUtils::printCargo(currentCargo, getMaxCargo(), os);
16     return os;
17 }
18
19 long double CargoShip::getCurrentCargo() const {
20     return currentCargo;
21 }
22
23
```

```
1 /**
2  * @file Implementation of the CargoShip class.
3 *
4  * @author Jallon Sarah
5  * @author Surdez Quentin
6 */
7 #pragma once
8
9 #include "src/ship/Ship.hpp"
10
11 /**
12  * @brief Abstract class representing a specialization of Ship that
13  * can carry cargo.
14  * they have a cargo capacity and a current cargo amount.
15 */
16
17 class CargoShip : public Ship {
18 private:
19     long double currentCargo;
20
21 protected:
22     /**
23      * @brief Constructor of CargoShip
24      * @throw std::invalid_argument if the current cargo is bigger than
25      * the max cargo
26      * @param _factoryNumber the factory number of the ship
27      * @param _maxCargo the max cargo capacity
28      * @param _currentCargo the current cargo amount
29      */
30     explicit CargoShip(size_t _factoryNumber, long double _maxCargo,
31                         long double _currentCargo);
32
33 public:
34     /**
35      * @brief Get the max cargo capacity
36      * @return the max cargo capacity
37      */
38     virtual long double getMaxCargo() const = 0;
39
40     /**
41      * @brief Get the current cargo amount.
42      * @return the current cargo amount.
43      */
44     long double getCurrentCargo() const;
45
46     /**
47      * @inheritDoc
48      */
49     std::ostream &formatToStream(std::ostream &os) const override;
50 }
```

51

52

53

```
1 #include "StarDreadnought.hpp"
2
3 size_t StarDreadnought::count = 1;
4
5 StarDreadnought::StarDreadnought(long double _currentCargo) :
6     CargoShip(count++, getMaxCargo(), _currentCargo) {
7 }
8
9 long double StarDreadnought::getWeight() const {
10     return 9e9 + getCurrentCargo();
11 }
12
13 unsigned long long int StarDreadnought::getSpeed() const {
14     return 40;
15 }
16
17 std::string StarDreadnought::getFactoryName() const {
18     return "Super-class Star Destroyer";
19 }
20
21 long double StarDreadnought::getMaxCargo() const {
22     return 250e3;
23 }
24
25
```

```
1 /**
2  * @file Implementation of the StarDreadnought class.
3 *
4  * @author Jallon Sarah
5  * @author Surdez Quentin
6 */
7 #pragma once
8
9 #include "src/ship/cargo/CargoShip.hpp"
10
11 /**
12  * @brief A class representing a star dreadnought. It is a
13  * specialization of the CargoShip class.
14 */
15 class StarDreadnought : public CargoShip {
16 private:
17     static size_t count;
18 public:
19
20     /**
21      * @inheritDoc
22      */
23     long double getWeight() const override;
24
25     /**
26      * @inheritDoc
27      */
28     unsigned long long int getSpeed() const override;
29
30     /**
31      * @inheritDoc
32      */
33     std::string getFactoryName() const override;
34
35     /**
36      * @inheritDoc
37      */
38     long double getMaxCargo() const override;
39
40     /**
41      * @brief Constructor of StarDreadnought
42      * @param _currentCargo the current cargo amount
43      */
44     explicit StarDreadnought(long double _currentCargo);
45
46 };
47
48
49
```

```
1
2 /**
3 * @file Implementation of the different tests we make on the classes
4 * we coded
5 *
6 * @author Jallon Sarah
7 * @author Surdez Quentin
8 */
9 #include <gtest/gtest.h>
10 #include "src/ship/TIE/TIE.hpp"
11 #include "src/ship/cargo/Shuttle.hpp"
12 #include "src/ship/cargo/StarDreadnought.hpp"
13 #include "../squadron/Squadron.hpp"
14 #include "src/ship/TIE/TIEInterceptor.hpp"
15 #include "../utils/ostreamUtils.hpp"
16
17 TEST(ConstructorSquadron, defaultConstructor) {
18     Squadron squadron("Squadron 1");
19     ASSERT_EQ(squadron.empty(), true);
20 }
21
22 TEST(ConstrucorSquadron, copyConstructor) {
23     Squadron squadron("Squadron 1");
24     TIEInterceptor interceptor;
25     squadron.addShip(interceptor);
26     TIEInterceptor interceptor1;
27     squadron.addShip(interceptor1);
28     Squadron squadron2(squadron);
29     std::cout << squadron << std::endl;
30     std::cout << squadron2 << std::endl;
31     ASSERT_EQ(squadron.empty(), false);
32     ASSERT_EQ(squadron2.empty(), false);
33 }
34
35 TEST(ConstructorSquadron, moveConstructor) {
36     Squadron squadron("Squadron 1");
37     TIEInterceptor interceptor;
38     squadron.addShip(interceptor);
39     TIEInterceptor interceptor1;
40     squadron.addShip(interceptor1);
41     Squadron squadron2(std::move(squadron));
42     std::cout << squadron << std::endl;
43     std::cout << squadron2 << std::endl;
44     ASSERT_EQ(squadron.empty(), true);
45     ASSERT_EQ(squadron2.empty(), false);
46 }
47
48 TEST(Assignments, CopyAssignment) {
49     Squadron squadron("Squadron 1");
50     TIEInterceptor interceptor;
51     squadron.addShip(interceptor);
52     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
```

```
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(),
184     squadron.getMaximumSpeed()), 119791.77);
185 }
186 TEST(Consumption, TotalConsumptionIfEmpty) {
187     Squadron squadron("Squadron 1");
188     EXPECT_THROW(
189         {
190             try {
191                 std::cout << squadron.fuelConsumption(squadron.
192                 getWeight(), squadron.getMaximumSpeed()) << std::endl;
193             } catch (std::runtime_error &re) {
194                 EXPECT_STREQ("Squadron is empty", re.what());
195                 throw;
196             }
197         }, std::runtime_error);
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
```

```
1  
2 #include <iomanip>  
3 #include "ostreamUtils.hpp"  
4  
5 std::ostream &ostreamUtils::printWeight(long double weight, std::  
    ostream &os) {  
6     fixed(os);  
7     os << std::setprecision(2) << "weight: " << weight << " tons\n";  
8     return os;  
9 }  
10  
11 std::ostream &ostreamUtils::printSpeed(unsigned long long speed, std::  
    ostream &os) {  
12     fixed(os);  
13     os << std::setprecision(0) << "max speed: " << speed << " MGLT\n";  
14     return os;  
15 }  
16  
17 std::ostream &ostreamUtils::printCargo(long double currentCargo, long  
    double maxCargo, std::ostream &os) {  
18     fixed(os);  
19     os << std::setprecision(1) << "cargo: " << currentCargo << " tons  
        ( max : " << maxCargo << ")\n";  
20     return os;  
21 }  
22  
23 std::ostream &ostreamUtils::fixed(std::ostream &os) {  
24     os << std::fixed;  
25     return os;  
26 }
```

```
1  /**
2   * @file Implementation of the namespace ostreamUtils
3   *
4   * @author Jallon Sarah
5   * @author Surdez Quentin
6   */
7  #pragma once
8
9 #include <iostream>
10
11 /**
12  * @brief A namespace containing functions to print information
13 */
14 namespace ostreamUtils {
15
16 /**
17  * @brief Print the weight with specific precision of 2
18  * @param weight the weight to print
19  * @param os the output stream
20  * @return the changed output stream
21  */
22 std::ostream &printWeight(long double weight, std::ostream &os);
23
24 /**
25  * @brief Print the speed with specific precision of 0
26  * @param speed the speed to print
27  * @param os the output stream
28  * @return the changed output stream
29  */
30 std::ostream &printSpeed(unsigned long long speed, std::ostream &os)
31 );
32 /**
33  * @brief Print the current cargo and the max cargo with specific
34  * precision of 1
35  * @param currentCargo the current cargo to print
36  * @param maxCargo the max cargo to print
37  * @param os the output stream
38  * @return the changed output stream
39  */
40 std::ostream &printCargo(long double currentCargo, long double
41 maxCargo, std::ostream &os);
42
43 /**
44  * @brief Set the output stream to fixed
45  * @param os the output stream
46  * @return the changed output stream
47  */
48 std::ostream &fixed(std::ostream &os);
49
50 }
```

```
1
2 #include "Squadron.hpp"
3 #include "src/utils/ostreamUtils.hpp"
4
5 Squadron operator+(const Squadron &rhs, Ship &lhs) {
6     return Squadron(rhs).addShip(lhs);
7 }
8
9 Squadron operator-(const Squadron &rhs, Ship &lhs) {
10    return Squadron(rhs).deleteShip(lhs);
11 }
12
13 std::ostream &operator<<(std::ostream &os, const Squadron &squadron) {
14     return squadron.formatToStream(os);
15 }
16
17 Squadron::Squadron(const std::string &_name) {
18     initSquadron(_name);
19 }
20
21 Squadron::~Squadron() {
22     deleteSquadron();
23 }
24
25 Squadron::Squadron(const Squadron &other) {
26     initSquadron(other.name, other.chief);
27     copySquadron(other);
28 }
29
30 Squadron::Squadron(Squadron &&other) noexcept {
31     initSquadron(other.name);
32     exchange(other);
33 }
34
35 Squadron &Squadron::operator=(const Squadron &other) {
36     if (this != &other) {
37         deleteSquadron();
38         initSquadron(other.name, other.chief);
39         copySquadron(other);
40     }
41     return *this;
42 }
43
44 Squadron &Squadron::operator=(Squadron &&other) noexcept {
45     if (this != &other) {
46         deleteSquadron();
47         initSquadron(other.name);
48         exchange(other);
49     }
50     return *this;
51 }
52
53 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         ) {
59             return *this;
60         }
61         current = current->next;
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 double Squadron::fuelConsumption(long double distance, unsigned long
143                                     long int speed) const {
144
145     if (distance < 0 || speed <= 0) {
146         throw std::invalid_argument("Distance and speed must be
147                                   positive");
148     }
149     if (speed > getMaximumSpeed()) {
150         throw std::invalid_argument("Speed cannot be higher than the
151                                   maximum speed at which the squadron travel");
152     }
153     if (empty()) {
154         throw std::runtime_error("Squadron is empty");
155 }
```

```
156
157     long double weight = getWeight();
158     return std::cbrtl(weight) / 2.0 * log10l(weight * static_cast<long
159     double>(speed)) * log10l(distance + 1.0);
160 }
161
162 unsigned long long int Squadron::getMaximumSpeed() const {
163     unsigned long long minSpeed = std::numeric_limits<unsigned long
164     long>::max();
165     Node *current = head;
166     while (current != nullptr) {
167         minSpeed = std::min(minSpeed, current->value->getSpeed());
168         current = current->next;
169     }
170     return minSpeed;
171 }
172 long double Squadron::getWeight() const {
173     long double totalWeight = 0.0;
174     Node *current = head;
175     while (current != nullptr) {
176         totalWeight += current->value->getWeight();
177         current = current->next;
178     }
179     return totalWeight;
180 }
181
182 std::ostream &Squadron::formatToStream(std::ostream &os) const {
183     if (!empty()) {
184         os << "Squadron " << name << "\n";
185         ostreamUtils::printSpeed(getMaximumSpeed(), os);
186         ostreamUtils::printWeight(getWeight(), os);
187         if (chief != nullptr) {
188             os << "--Leader:\n" << *chief;
189         }
190         if (head->value == chief && head->next != nullptr || head->
191             value != chief) {
192             os << "--Members:\n";
193             Node *current = head;
194             while (current != nullptr) {
195                 if (current->value == chief) {
196                     current = current->next;
197                     continue;
198                 }
199                 os << *current->value;
200                 current = current->next;
201             }
202         } else {
203             os << "Squadron is empty\n";
204         }
205     }
206     return os;
207 }
```

```
206 }
207
208 // Private methods
209
210 void Squadron::initSquadron(const std::string &n, pointer _chief) {
211     chief = _chief;
212     name = n;
213 }
214
215
216 void Squadron::deleteSquadron() {
217     Node *current = head;
218     while (current != nullptr) {
219         Node *next = current->next;
220         delete current;
221         current = next;
222     }
223 }
224
225 void Squadron::copySquadron(const Squadron &other) {
226     if (head == other.head) {
227         return;
228     }
229     for (size_t i = 0; i < other.size; i++) {
230         addShip(other.get(i));
231     }
232 }
233
234 void Squadron::exchange(Squadron &other) {
235     head = std::exchange(other.head, nullptr);
236     name = std::exchange(other.name, std::string{});
237     chief = std::exchange(other.chief, nullptr);
238     size = std::exchange(other.size, 0);
239 }
240
241 bool Squadron::contains(Squadron::reference ship) {
242     Node *current = head;
243     while (current != nullptr) {
244         if (current->value == &ship) {
245             return true;
246         }
247         current = current->next;
248     }
249     return false;
250 }
251
```

```

1 /**
2  * @file Implementation of the Squadron class.
3 *
4  * @author Jallon Sarah
5  * @author Surdez Quentin
6 */
7 #pragma once
8
9 #include <string>
10 #include "src/ship/Ship.hpp"
11
12 class Squadron;
13
14 /**
15  * @brief Implement the operator+ to add a ship to a squadron
16  * @param rhs the right hand side of the operator which is a squadron
17  * @param lhs the left hand side of the operator which is a ship
18  * @return a new squadron with the ship added
19 */
20 [[nodiscard]] Squadron operator+(const Squadron &rhs, Ship &lhs);
21
22 /**
23  * @brief Implement the operator- to remove a ship from a squadron
24  * @param rhs the right hand side of the operator which is a squadron
25  * @param lhs the left hand side of the operator which is a ship
26  * @return a new squadron with the ship removed
27 */
28 [[nodiscard]] Squadron operator-(const Squadron &rhs, Ship &lhs);
29
30 /**
31  * @brief Implement the operator<< to format the squadron to an output
32  * stream
33  * @param os the output stream of interest
34  * @param squadron the squadron to format
35  * @return the output stream changed
36 */
37 std::ostream &operator<<(std::ostream &os, const Squadron &squadron);
38
39 /**
40  * @brief Squadron class is used to represent a squadron of ships. It
41  * is a linked list of ships.
42  * Each squadron has a name, a potential chief and a list of ships
43  * that are a part of the
44  * squadron. The rule of five has been followed to ensure that the
45  * class corresponds to
46  * the extended canonical form.
47 */
48
49 class Squadron {
50 public:
51     using value_type = Ship;
52     using pointer = Ship *;
53     using reference = Ship &;
54     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
129      * ship
130      * @param lhs the left hand side of the operator which is a ship
131      * @return the instance of the squadron with the ship removed
132     */
133     Squadron &operator-=(reference lhs);
134
135     /**
136      * @brief Sets chief to the squadron. If the chief is not a part
137      * of the squadron, it is added to the squadron.
138      * @param newChief the ship to set as the chief
139     */
140     void setChief(reference newChief);
141
142     /**
143      * @brief Resets the chief of the squadron. Does not remove the
144      * ship from the squadron.
145     */
146     void removeChief();
147
148     /**
149      * @brief Get the i-th ship of the squadron
150      * @throw std::out_of_range if i is out of the index
151      * @param i index of the ship of interest
152      * @return a reference on the ship of interest
153     */
154     [[nodiscard]] reference get(size_t i) const;
```

```

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::runtime_error if Squadron is empty
169  * @throw std::invalid_argument if distance or speed is negative
170  * or if speed is greater than the maximum speed
171  * @param distance the distance the squadron will travel
172  * @param speed the speed at which the squadron will travel
173  * @return the estimate consumption for the whole squadron
174  */
175   double fuelConsumption(long double distance, unsigned long long
176   int speed) const;
177
178 /**
179  * @brief Get the maximum speed at which the squadron can travel
180  * @return the maximum speed at which the squadron can travel
181  */
182   unsigned long long int getMaximumSpeed() const;
183
184 /**
185  * @brief Get the total weight of the squadron
186  * @return the total weight of the squadron
187  */
188   long double getWeight() const;
189
190 /**
191  * @brief Format the squadron for an output stream
192  * @param os the output stream of interest
193  * @return the changed output stream
194  */
195   std::ostream &formatToStream(std::ostream &os) const;
196
197 private:
198
199   struct Node {
200     pointer value;
201     Node *next;
202   };
203
204   Node *head{};
205   std::string name;
206   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
```