Airlines

Description of the project

The project implements the structure of airline companies. The leading class company holds all their flights in a list and references to owned planes, employed stewardesses, and pilots (crew members). Plane, flight, and crew member store company pointers to forbid assigning them to multiple airline companies.

Passengers, crew members, and planes are stored in a list outside any class; on the other hand, flights are stored inside the company they belong to in the list. They are created and destroyed using the function from the company class (flight constructors are private).

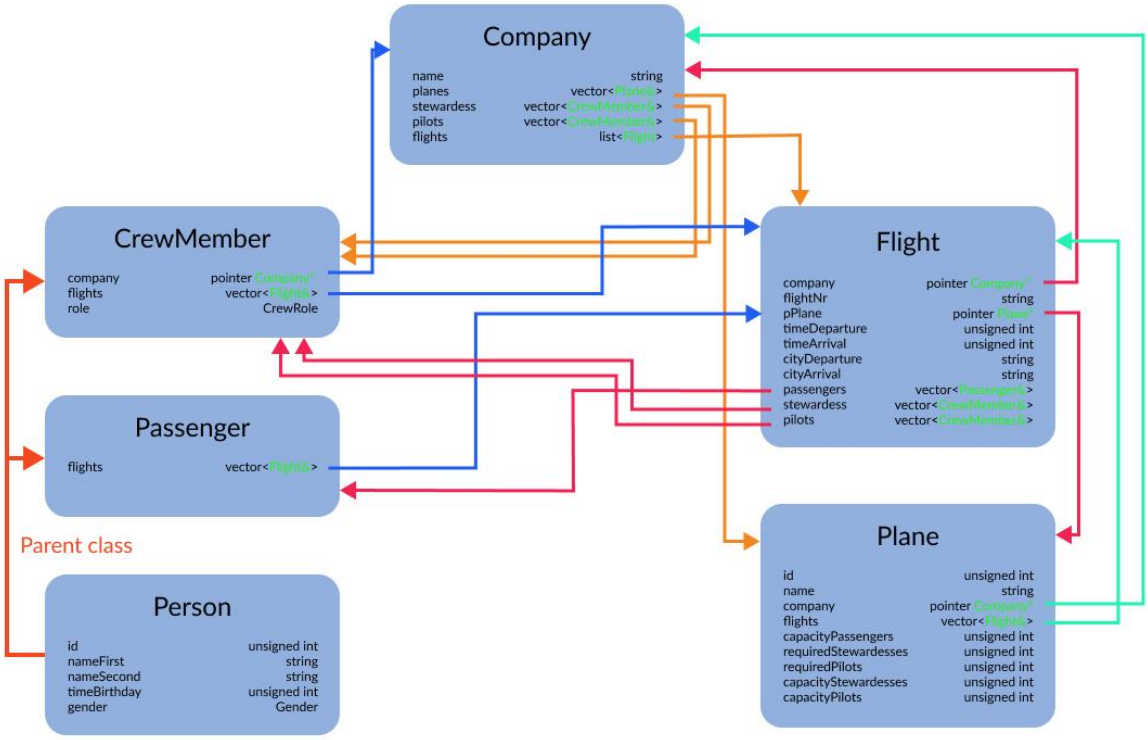
Each flight consists of the plane, passenger, stewardess, pilots, company to which it belongs, and characteristic data for the flight: city of departure, arrival, and UTC of departure and arrival in milliseconds.

The class plane includes all of its flights, ID, name, company that belongs to, and parameters describing the plane, such as passengers, stewardesses, pilots capacity, and required number of stewardesses and pilots to operate the flight.

The next class is a person with ID, first and second name, birthday time (UTC in milliseconds), and gender. Class person is parent class for passenger and crew member classes.

The passenger class contains flights. The crew member class contains flights and the role of the crew member: pilot or stewardess. Separating these classes will allow the implementation of specific functions and parameters for these classes in the future.

Case study



Declaration of the classes

```
class Company {
   public:
    Company(std::string name);

    ~Company();

    void setName(std::string name);

    std::list<Flight>& getFlights();
    std::vector<std::reference_wrapper<Plane>>& getPlanes();
    std::vector<std::reference_wrapper<CrewMember>>& getStewardesses();
    std::vector<std::reference_wrapper<CrewMember>>& getPilots();
    // ADD PLANE TO COMPANY AND SET PLANE'S COMPANY
    void addPlane(Plane& plane);
```

```
    // REMOVE PLANE FROM COMPANY AND REMOVES PLANE'S COMPANY
    bool removePlane(Plane& plane);
    bool removePlanes();

    // ADD CREW MEMBER TO COMPANY AND SET CREW MEMBER'S COMPANY
    void addCrewMember(CrewMember& crewMember);
    // REMOVE CREW MEMBER FROM COMPANY AND REMOVES CREW MEMBER'S COMPANY
    bool removeCrewMember(CrewMember& crewMember);
    bool removeCrewMembers();

    // CRETE FLIGHT IN COMPANY
    Flight& createFlight(std::string flightNr,
                         Plane& plane,
                         unsigned int timeDeparture,
                         unsigned int timeArrival,
                         std::string cityDeparture,
                         std::string cityArrival);

    Flight& createFlight(std::string flightNr,
                         unsigned int timeDeparture,
                         unsigned int timeArrival,
                         std::string cityDeparture,
                         std::string cityArrival);
    // REMOVE FLIGHT, DELETE FROM ALL CLASSES
    bool removeFlight(Flight& flight);
    bool removeFlights();

   private:
    std::string name;
    std::list<Flight> flights;
    std::vector<std::reference_wrapper<Plane>> planes;
    std::vector<std::reference_wrapper<CrewMember>> stewardesses;
    std::vector<std::reference_wrapper<CrewMember>> pilots;
};
```

```cpp
class Plane {
   public:
    static std::vector<unsigned int> usedIds;

    Plane(
        Company* pCompany,
        unsigned int id,
        std::string name,
        unsigned int capacityPassengers,
        unsigned int requiredStewardesses,
        unsigned int requiredPilots);

    Plane(
        unsigned int id,
        std::string name,
        unsigned int capacityPassengers,
        unsigned int requiredStewardesses,
        unsigned int requiredPilots);

    ~Plane();

    unsigned int getId() const;
    std::string getName() const;
    Company*& getCompany();
    std::vector<std::reference_wrapper<Flight>>& getFlights();
    unsigned int getCapacityPassengers() const;
    unsigned int getRequiredStewardesses() const;
    unsigned int getCapacityStewardesses() const;
    unsigned int getRequiredPilots() const;
    unsigned int getCapacityPilots() const;

    void setCompany(Company* pCompany);
```

```
    void changeId(const unsigned int id);
    void setName(const std::string name);

    void setCapacityPassengers(const unsigned int number);
    void setRequiredStewardesses(const unsigned int number);
    void setRequiredPilots(const unsigned int number);

    bool inRangePassengers(const unsigned int number) const;

    bool inRangeStewardesses(const unsigned int number) const;
    bool maximumStewardesses(const unsigned int number) const;

    bool inRangePilots(const unsigned int number) const;
    bool maximumPilots(const unsigned int number) const;

    bool inRangeCrew(const unsigned int stewardess, const unsigned int
pilots) const;

    // INVOKE FUNCTION FROM FLIGHT TO ADD PLANE
    void addFlight(Flight& flight);
    // INVOKE FUNCTION FROM FLIGHT TO REMOVE PLANE
    bool removeFlight(Flight& flight);
    // INVOKE FOR ALL FLIGHTS FUNCTION removeFlight
    bool removeFlights();

   private:
    void setId(const unsigned int id);

    unsigned int id;
    std::string name;

    Company* pCompany;
    std::vector<std::reference_wrapper<Flight>> flights;
```

```
    unsigned int capacityPassengers;
    unsigned int requiredStewardesses;
    unsigned int requiredPilots;

    unsigned int capacityStewardesses;
    unsigned int capacityPilots;
};
```

```
class Flight {
   public:
    Company* getCompany();
    std::string getFlightNr() const;
    FlightStatus getStatus() const;
    Plane* getPlane() const;
    unsigned int getTimeDeparture() const;
    unsigned int getTimeArrival() const;
    std::string getCityDeparture() const;
    std::string getCityArrival() const;

    std::vector<std::reference_wrapper<Passenger>>& getPassengers();
    std::vector<std::reference_wrapper<CrewMember>>& getStewardesses();
    std::vector<std::reference_wrapper<CrewMember>>& getPilots();

    void setFlightNr(const std::string flightNr);
    // CHANGE PLANE FOR ANOTHER AND CHECK IF PASSENGERS, STEWARDESS, PILOTS
ARE WITHIN RANGE
    void setPlane(Plane& plane);
    bool removePlane();

    // CHANGE DATA TIME CALLED IN CONSTRUCTOR VALIDATING ARGUMENTS WITH THIS
DATA PARAMETERS
    void changeDataDeparture(const unsigned int time);
    void changeDataArrival(const unsigned int time);
```

```
    void changeDataDeparture(const unsigned int time, const std::string
city);
    void changeDataArrival(const unsigned int time, const std::string city);

    // ADD PASSENGER TO FLIGHT,ADD FLIGHT TO PASSENGER
    void addPassenger(Passenger& passenger);
    // REMOVE PASSENGER FROM FLIGHT, REMOVE FLIGHT FROM PASSENGER
    bool removePassenger(Passenger& passenger);
    bool removePassengers();

    // ADD CREW MEMBER TO FLIGHT, ADD FLIGHT TO CREW MEMBER
    void addCrewMember(CrewMember& crewMember);
    // REMOVE CREW MEMBER FROM FLIGHT, REMOVE FLIGHT FROM CREW MEMBER
    bool removeCrewMember(CrewMember& crewMember);
    bool removeCrewMembers();

    // CHECK FOR FLIGHT TIME OVERLAP WITH TIME PERIOD
    bool timeOverlap(const Flight& flight) const;
    bool timeOverlap(const unsigned int timeStart, const unsigned int
timeEnd) const;

   private:
    Flight(
        Company* pCompany,
        std::string flightNr,
        unsigned int timeDeparture,
        unsigned int timeArrival,
        std::string cityDeparture,
        std::string cityArrival);

    void setStatus();
    void setupCompany(Company* pCompany);
    // SET PLANE CALLED IN CONSTRUCTOR WITHOUT ANY CHECK ON CAPACITY LIMITS
    void setupPlane(Plane& plane);
```

```
    // SET DATA TIME CALLED IN CONSTRUCTOR VALIDATING ARGUMENTS
    void setDataTime(const unsigned int timeDeparture, const unsigned int
timeArrival);
    void setDataTime(const unsigned int timeDeparture, const std::string
cityDeparture, const unsigned int timeArrival, const std::string
cityArrival);

    Company* pCompany;
    std::string flightNr;
    FlightStatus status;
    Plane* pPlane;

    unsigned int timeDeparture;  // time in miliseconds
    unsigned int timeArrival;    // time in miliseconds
    std::string cityDeparture;
    std::string cityArrival;

    std::vector<std::reference_wrapper<Passenger>> passengers;
    std::vector<std::reference_wrapper<CrewMember>> stewardesses;
    std::vector<std::reference_wrapper<CrewMember>> pilots;

    friend Company;
};
```

```cpp
class Person {
   public:
    static std::vector<unsigned int> usedIds;

    Person(
        unsigned int id,
        std::string nameFirst,
        std::string nameSecond,
        unsigned int timeBirthday,
        Gender gender);

    ~Person();

    unsigned int getId() const;
    std::string getNameFirst() const;
    std::string getNameSecond() const;
    unsigned int getTimeBirthday() const;
    Gender getGender() const;

    void changeId(const unsigned int id);
    void setNameFirst(const std::string name);
    void setNameSecond(const std::string name);

    void setTimeBirthday(const unsigned int time);

   private:
    void setId(const unsigned int id);

    unsigned int id;
    std::string nameFirst;
    std::string nameSecond;
    unsigned int timeBirthday;  // time in miliseconds UTC
    Gender gender;
};
```

```
class Passenger : public Person {
   public:
    Passenger(
        unsigned int id,
        std::string nameFirst,
        std::string nameSecond,
        unsigned int timeBirthday,
        Gender gender);

    ~Passenger();

    std::vector<std::reference_wrapper<Flight>>& getFlights();

    // INVOKE FUNCTION FROM FLIGHT TO ADD PASSENGER
    void addFlight(Flight& flight);
    // INVOKE FUNCTION FROM FLIGHT TO REMOVE PASSENGER
    bool removeFlight(Flight& flight);

    // INVOKE FOR ALL FLIGHTS FUNCTION removeFlight
    bool removeFlights();

   private:
    std::vector<std::reference_wrapper<Flight>> flights;
};
```

```
class CrewMember : public Person {
   public:
    CrewMember(
        Company* pCompany,
        CrewRole role,
        unsigned int id,
        std::string nameFirst,
        std::string nameSecond,
```

```
        unsigned int timeBirthday,
        Gender gender);

    CrewMember(
        CrewRole role,
        unsigned int id,
        std::string nameFirst,
        std::string nameSecond,
        unsigned int timeBirthday,
        Gender gender);

    ~CrewMember();

    Company*& getCompany();
    std::vector<std::reference_wrapper<Flight>>& getFlights();
    CrewRole getRole() const;

    // INVOKE FUNCTION FROM COMPANY TO ADD CREW MEMBER
    void setCompany(Company* pCompany);

    // INVOKE FUNCTION FROM FLIGHT TO ADD CREW MEMBER
    void addFlight(Flight& flight);
    // INVOKE FUNCTION FROM FLIGHT TO REMOVE CREW MEMBER
    bool removeFlight(Flight& flight);

    // INVOKE FOR ALL FLIGHTS FUNCTION removeFlight
    bool removeFlights();

  private:
    Company* pCompany;
    std::vector<std::reference_wrapper<Flight>> flights;

    CrewRole role;
};
```

Tests Example

```
TEST(crewMember, company) {
    Company* pCompany1 = new (Company){"Test1"};
    Company* pCompany2 = new (Company){"Test2"};

    CrewMember crewMember{PILOT, 1, "Victor", "Alb", 1, MALE};
    // setCompany
    // SETTING NULLPTR, PREVIOUS NULLPTR
    EXPECT_NO_THROW(crewMember.setCompany(nullptr));
    EXPECT_EQ(crewMember.getCompany(), nullptr);
    // SETTING COMPANY, PREVIOUS NULLPTR
    crewMember.setCompany(pCompany1);

    EXPECT_EQ(crewMember.getCompany(), pCompany1);
    EXPECT_EQ(pCompany1->getPilots().size(), 1);
    EXPECT_TRUE(existVector(pCompany1->getPilots(), crewMember));
    // SETTING COMPANY, PREVIOUS COMPANY
    crewMember.setCompany(pCompany2);

    EXPECT_EQ(pCompany1->getPilots().size(), 0);

    EXPECT_EQ(crewMember.getCompany(), pCompany2);
    EXPECT_EQ(pCompany2->getPilots().size(), 1);
    EXPECT_TRUE(existVector(pCompany2->getPilots(), crewMember));
    // SETTING NULLPTR, PREVIOUS COMPANY
    crewMember.setCompany(nullptr);

    EXPECT_EQ(pCompany2->getPilots().size(), 0);
    EXPECT_EQ(crewMember.getCompany(), nullptr);

    delete pCompany1, pCompany2;
}
```

```
TEST(crewMember, flight) {
    Company* pCompany = new (Company){"Test1"};
    Flight& flight1 = pCompany->createFlight("RYR123", 1, 2, "Warsaw",
"Berlin");
    Flight& flight2 = pCompany->createFlight("RYR123", 3, 4, "Warsaw",
"Berlin");

    CrewMember crewMember{pCompany, PILOT, 1, "Victor", "Alb", 1, MALE};
    // addFlight
    crewMember.addFlight(flight1);

    EXPECT_EQ(crewMember.getFlights().size(), 1);
    EXPECT_EQ(flight1.getPilots().size(), 1);

    EXPECT_TRUE(existVector(crewMember.getFlights(), flight1));
    EXPECT_TRUE(existVector(flight1.getPilots(), crewMember));
    // removeFlight
    crewMember.addFlight(flight2);
    EXPECT_TRUE(crewMember.removeFlight(flight1));

    EXPECT_EQ(crewMember.getFlights().size(), 1);
    EXPECT_EQ(flight1.getPilots().size(), 0);
    EXPECT_EQ(flight2.getPilots().size(), 1);

    EXPECT_FALSE(existVector(flight1.getPilots(), crewMember));
    EXPECT_FALSE(existVector(crewMember.getFlights(), flight1));
    EXPECT_TRUE(existVector(crewMember.getFlights(), flight2));
    // removeFlights
    crewMember.addFlight(flight1);
    EXPECT_TRUE(crewMember.removeFlights());

    EXPECT_EQ(crewMember.getFlights().size(), 0);
    EXPECT_EQ(flight1.getPilots().size(), 0);
```

```
    EXPECT_FALSE(existVector(crewMember.getFlights(), flight1));
    EXPECT_FALSE(existVector(flight1.getPilots(), crewMember));

    EXPECT_FALSE(existVector(crewMember.getFlights(), flight2));
    EXPECT_FALSE(existVector(flight2.getPilots(), crewMember));

    delete pCompany;
}
```