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 has sets of references to owned planes, employed crew members (stewardesses and pilots). 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 functions from the company class (flight constructors are private).

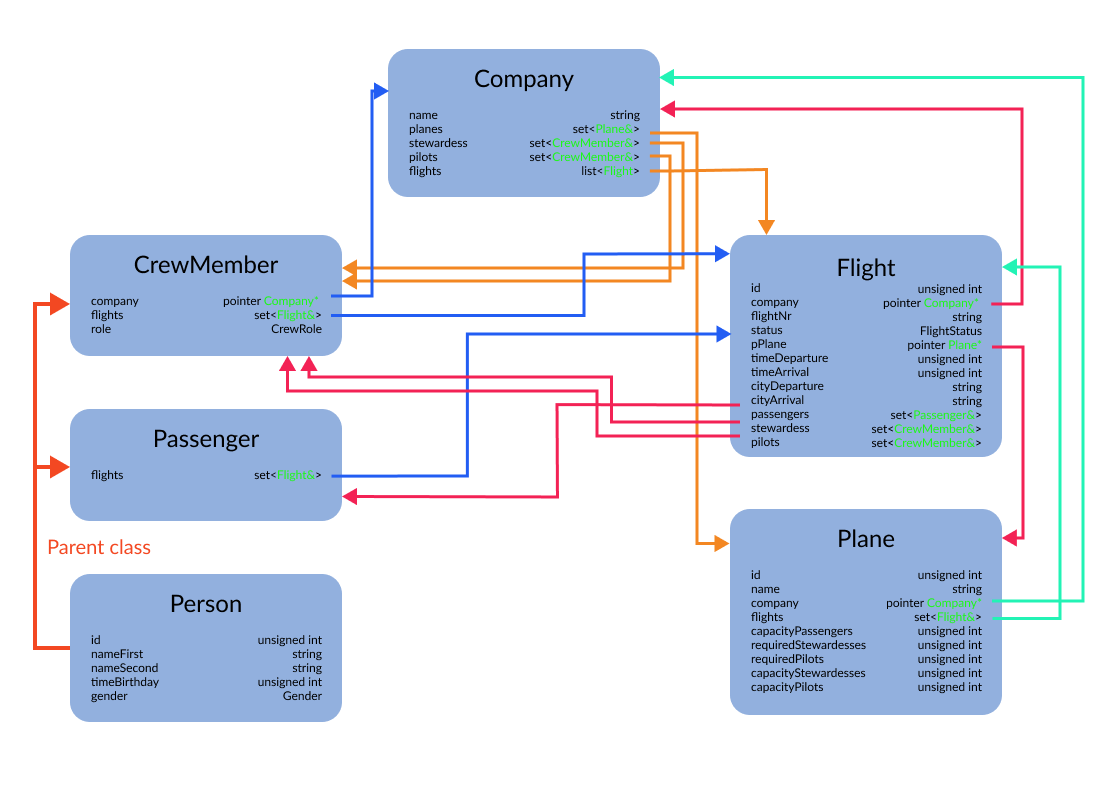
Each flight consists of the ID, flight status, sets of reference to plane, passenger, stewardess, pilots. Pointer to company to which belongs, and characteristic data for the flight: city of departure, arrival, and UTC of departure and arrival in milliseconds.

The class plane has set of reference to all 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 and crew member class contains set of reference to flights. The crew member class is more extended as it store 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();

    std::string getName() const;

    std::list<Flight>& getFlights();

    std::set<std::reference\_wrapper<Plane>>& getPlanes();

    std::set<std::reference\_wrapper<CrewMember>>& getStewardesses();

    std::set<std::reference\_wrapper<CrewMember>>& getPilots();

    void setName(const std::string name);

*// 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 long long timeDeparture,

                         unsigned long long timeArrival,

                         std::string cityDeparture,

                         std::string cityArrival);

    Flight& createFlight(std::string flightNr,

                         unsigned long long timeDeparture,

                         unsigned long long timeArrival,

                         std::string cityDeparture,

                         std::string cityArrival);

*// REMOVE FLIGHT, DELETE FROM ALL CLASSES*

    bool removeFlight(Flight& flight);

    bool removeFlights();

    std::set<std::reference\_wrapper<CrewMember>> availableCrewMembers(unsigned long long timeStart, unsigned long long timeEnd, CrewRole role);

    friend std::ostream& operator<<(std::ostream& os, Company& company);

   private:

    std::string name;

    std::list<Flight> flights;

    std::set<std::reference\_wrapper<Plane>> planes;

    std::set<std::reference\_wrapper<CrewMember>> stewardesses;

    std::set<std::reference\_wrapper<CrewMember>> pilots;

};

class Person {

   public:

    static std::vector<unsigned int> usedIds;

    Person(

        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 setNameFirst(const std::string name);

    void setNameSecond(const std::string name);

    void setTimeBirthday(const unsigned int time);

   private:

    void setId();

    unsigned int id;

    std::string nameFirst;

    std::string nameSecond;

    unsigned int timeBirthday; *// time in miliseconds UTC*

    Gender gender;

};

class CrewMember : public Person {

   public:

    CrewMember(

        Company\* pCompany,

        CrewRole role,

        std::string nameFirst,

        std::string nameSecond,

        unsigned int timeBirthday,

        Gender gender);

    CrewMember(

        CrewRole role,

        std::string nameFirst,

        std::string nameSecond,

        unsigned int timeBirthday,

        Gender gender);

    ~CrewMember();

    CrewRole getRole() const;

    Company\* getCompany() const;

    std::set<std::reference\_wrapper<Flight>>& getFlights();

*// INVOKE FUNCTION FROM COMPANY TO ADD CREW MEMBER*

    bool 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();

    friend bool operator==(const std::reference\_wrapper<CrewMember>& one, const CrewMember& other);

    friend bool operator<(const std::reference\_wrapper<CrewMember>& one, const std::reference\_wrapper<CrewMember>& other);

    friend std::ostream& operator<<(std::ostream& os, CrewMember& crewMember);

   private:

    Company\* pCompany;

    std::set<std::reference\_wrapper<Flight>> flights;

    CrewRole role;

};

class Passenger : public Person {

   public:

    Passenger(

        std::string nameFirst,

        std::string nameSecond,

        unsigned int timeBirthday,

        Gender gender);

    ~Passenger();

    std::set<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();

    friend bool operator==(const std::reference\_wrapper<Passenger>& one, const Passenger& other);

    friend bool operator<(const std::reference\_wrapper<Passenger>& one, const std::reference\_wrapper<Passenger>& other);

    friend std::ostream& operator<<(std::ostream& os, Passenger& passenger);

   private:

    std::set<std::reference\_wrapper<Flight>> flights;

};

class Plane {

   public:

    static std::vector<unsigned int> usedIds;

    Plane(

        Company\* pCompany,

        std::string name,

        unsigned int capacityPassengers,

        unsigned int requiredStewardesses,

        unsigned int requiredPilots);

    Plane(

        std::string name,

        unsigned int capacityPassengers,

        unsigned int requiredStewardesses,

        unsigned int requiredPilots);

    ~Plane();

    unsigned int getId() const;

    std::string getName() const;

    Company\* getCompany() const;

    unsigned int getCapacityPassengers() const;

    unsigned int getRequiredStewardesses() const;

    unsigned int getCapacityStewardesses() const;

    unsigned int getRequiredPilots() const;

    unsigned int getCapacityPilots() const;

    std::set<std::reference\_wrapper<Flight>>& getFlights();

    bool setCompany(Company\* pCompany);

    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();

    friend bool operator==(const std::reference\_wrapper<Plane>& one, const Plane& other);

    friend bool operator<(const std::reference\_wrapper<Plane>& one, const std::reference\_wrapper<Plane>& other);

    friend std::ostream& operator<<(std::ostream& os, Plane& plane);

   private:

    void setId();

    unsigned int id;

    std::string name;

    Company\* pCompany;

    std::set<std::reference\_wrapper<Flight>> flights;

    unsigned int capacityPassengers;

    unsigned int requiredStewardesses;

    unsigned int requiredPilots;

    unsigned int capacityStewardesses;

    unsigned int capacityPilots;

};

class Flight {

   public:

    ~Flight();

    static std::vector<unsigned int> usedIds;

    Company\* getCompany() const;

    unsigned int getId() const;

    std::string getFlightNr() const;

    FlightStatus getStatus() const;

    Plane\* getPlane() const;

    unsigned long long getTimeDeparture() const;

    unsigned long long getTimeArrival() const;

    std::string getCityDeparture() const;

    std::string getCityArrival() const;

    std::set<std::reference\_wrapper<Passenger>>& getPassengers();

    std::set<std::reference\_wrapper<CrewMember>>& getStewardesses();

    std::set<std::reference\_wrapper<CrewMember>>& getPilots();

    void setStatus();

    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 long long time);

    void changeDataArrival(const unsigned long long time);

    void changeDataDeparture(const unsigned long long time, const std::string city);

    void changeDataArrival(const unsigned long long 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 long long timeStart, const unsigned long long timeEnd) const;

    bool operator==(const Flight& other) const;

    friend bool operator==(const std::reference\_wrapper<Flight>& one, const Flight& other);

    friend bool operator<(const std::reference\_wrapper<Flight>& one, const std::reference\_wrapper<Flight>& other);

    friend std::ostream& operator<<(std::ostream& os, Flight& flight);

   private:

    Flight(

        Company\* pCompany,

        std::string flightNr,

        unsigned long long timeDeparture,

        unsigned long long timeArrival,

        std::string cityDeparture,

        std::string cityArrival);

    void setId();

*// 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 long long timeDeparture, const unsigned long long timeArrival);

    void setDataTime(const unsigned long long timeDeparture, const std::string cityDeparture, const unsigned long long timeArrival, const std::string cityArrival);

    unsigned int id;

    Company\* pCompany;

    std::string flightNr;

    FlightStatus status;

    Plane\* pPlane;

    unsigned long long timeDeparture; *// time in miliseconds*

    unsigned long long timeArrival; *// time in miliseconds*

    std::string cityDeparture;

    std::string cityArrival;

    std::set<std::reference\_wrapper<Passenger>> passengers;

    std::set<std::reference\_wrapper<CrewMember>> stewardesses;

    std::set<std::reference\_wrapper<CrewMember>> pilots;

    friend Company;

};

Tests

