Airlines

# Description of the project

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

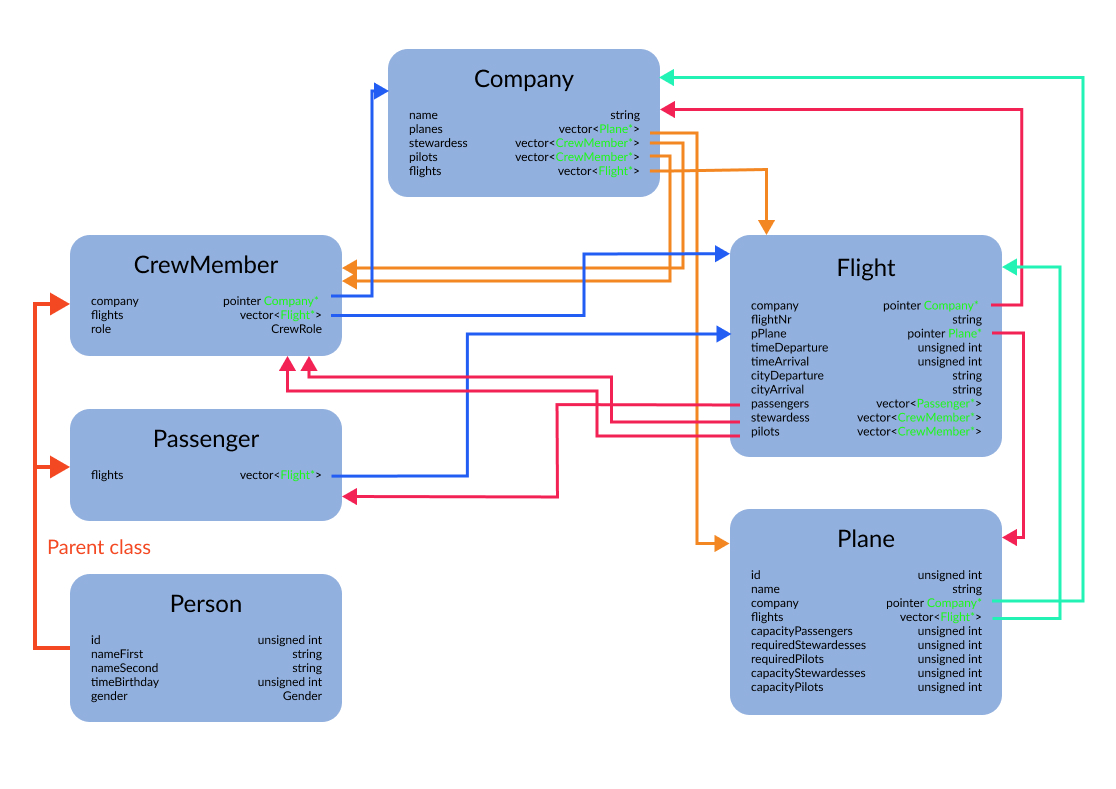
Each flight consists of the plane, passenger, stewardess, pilots, company to which it belongs, and characteristic data for the flight: city of departure, arrival, and UTC of departure and arrival in milliseconds.

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

## Company

class Company {

   public:

    Company(std::string name);

    void setName(std::string name);

    void addPlane(Plane\* pPlane);

    bool removePlane(Plane\* pPlane);

    void addFlight(Flight\* pFlight);

    bool removeFlight(Flight\* pFlight);

    void addStewardess(CrewMember\* pStewardess);

    bool removeStewardess(CrewMember\* pStewardess);

    void addPilot(CrewMember\* pPilot);

    bool removePilot(CrewMember\* pPilot);

*// DELETE ALL FLIGHTS, COMPANY'S PLANES AND MAKE CREW MEMBERS UNEMPLOYED*

    void terminate();

   private:

    std::string name;

    std::vector<Plane\*> planes;

    std::vector<Plane\*> flights;

    std::vector<CrewMember\*> stewardesses;

    std::vector<CrewMember\*> pilots;

};

## Plane

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;

    unsigned int getCapacityPassengers() const;

    unsigned int getRequiredStewardesses() const;

    unsigned int getCapacityStewardesses() const;

    unsigned int getRequiredPilots() const;

    unsigned int getCapacityPilots() const;

    void setCompany(Company\* pCompany);

*// SET ID AND REMOVE OLD ONE FROM VECTOR usedIds*

    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);

*// CHECK IF NUMBER OF PASSENGERS IS WITHIN CAPACITY RANGE*

    bool inRangePassengers(const unsigned int number) const;

    bool inRangePassengers(const std::vector<Passenger\*>& passengers) const;

*// CHECK IF NUMBER OF STEWARDESSES IS BETWEEN REQUIRED AND CAPACITY NUMBER*

    bool inRangeStewardesses(const unsigned int number) const;

    bool inRangeStewardesses(const std::vector<CrewMember\*>& stewardesses) const;

*// CHECK IF NUMBER OF PILOTS IS BETWEEN REQUIRED AND CAPACITY NUMBER*

    bool inRangePilots(const unsigned int number) const;

    bool inRangePilots(const std::vector<CrewMember\*>& pilots) const;

*// CHECK IN RANGE STEWARDESSES AND PILOTS*

    bool inRangeCrew(const std::vector<CrewMember\*>& stewardesses, const std::vector<CrewMember\*>& pilots) const;

*// ADD FLIGHT POINTER TO PLANE, IF PLANE IS NOT ADDED IN FLIGHT, ADD PLANE POINTER TO FLIGHT*

    void addFlight(Flight\* pFlight);

*// REMOVE FLIGHT POINTER FROM PLANE, IF PLANE IS ADDED IN FLIGHT, REMOVE PLANE POINTER FROM FLIGHT*

    void removeFlight(Flight\* pFlight);

*// REMOVE ALL FLIGHTS FROM PLANE AND PLANE FROM FLIGHTS*

    void removeFlights();

   private:

*// SET ID, CALLED IN CONSTRUCTOR*

    void setId(const unsigned int id);

    unsigned int id;

    std::string name;

    Company\* pCompany;

    std::vector<Flight\*> flights;

    unsigned int capacityPassengers;

    unsigned int requiredStewardesses;

    unsigned int requiredPilots;

    unsigned int capacityStewardesses;

    unsigned int capacityPilots;

};

## Person

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;

*// SET ID AND REMOVE OLD ONE FROM VECTOR usedIds*

    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:

*// SET ID, CALLED IN CONSTRUCTOR*

    void setId(const unsigned int id);

    unsigned int id;

    std::string nameFirst;

    std::string nameSecond;

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

    Gender gender;

};

## Passenger

class Passenger : public Person {

   public:

    Passenger(

        unsigned int id,

        std::string nameFirst,

        std::string nameSecond,

        unsigned int timeBirthday,

        Gender gender);

*// ADD FLIGHT POINTER TO PASSENGER, IF PASSENGER IS NOT ON THE FLIGHT, INVOKE FUNCTION IN FLIGHT TO ADD PASSENGER POINTER*

    void addFlight(Flight\* pFlight);

*// REMOVE FLIGHT POINTER FROM PASSENGER, IF PASSENGER IS ON THE FLIGHT, INVOKE FUNCTION IN FLIGHT TO REMOVE PASSENGER POINTER*

    bool removeFlight(Flight\* pFlight);

*// REMOVE ALL FLIGHTS FROM PASSENGER AND PASSENGER POINTERS FROM FLIGHTS*

    bool removeFlights();

*// CHECK IF PASSENGER IS ON THE FLIGHT*

    bool existFlight(Flight\* pFlight);

   private:

    std::vector<Flight\*> flights;

};

## CrewMember

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);

    void setCompany(Company\* pCompany);

*// ADD FLIGHT POINTER TO CREW MEMBER, IF CREW MEMBER IS NOT ON THE FLIGHT, INVOKE FUNCTION IN FLIGHT TO ADD CREW MEMBER POINTER TO FLIGHT*

    void addFlight(Flight\* pFlight);

*// REMOVE FLIGHT POINTER FROM CREW MEMBER, IF CREW MEMBER IS ON THE FLIGHT, INVOKE FUNCTION IN FLIGHT TO REMOVE CREW MEMBER POINTER*

    void removeFlight(Flight\* pFlight);

*// REMOVE ALL FLIGHTS FROM CREW MEMBER AND CREW MEMBER POINTERS FROM FLIGHTS*

    bool removeFlights();

*// CALLED IN DESTRUCTOR REMOVE ALL FLIGHTS AND CREW MEMBER FROM COMPANY*

    void terminate();

   private:

    Company\* pCompany;

    std::vector<Flight\*> flights;

    CrewRole role;

};

## Flight

class Flight {

   public:

    Flight(

        Company\* pCompany,

        std::string flightNr,

        Plane\* pPlane,

        unsigned int timeDeparture,

        unsigned int timeArrival,

        std::string cityDeparture,

        std::string cityArrival);

    Flight(

        Company\* pCompany,

        std::string flightNr,

        unsigned int timeDeparture,

        unsigned int timeArrival,

        std::string cityDeparture,

        std::string cityArrival);

    Flight(

        std::string flightNr,

        unsigned int timeDeparture,

        unsigned int timeArrival,

        std::string cityDeparture,

        std::string cityArrival);

    std::string getFlightNr() const;

    Plane\* getPlane() const;

    unsigned int getTimeDeparture() const;

    unsigned int getTimeArrival() const;

    std::string getCityDeparture() const;

    std::string getCityArrival() const;

    void setCompany(Company\* pCompany);

    void setFlightNr(const std::string flightNr);

*// CHANGE PLANE FOR ANOTHER AND CHECK IF PASSENGERS, STEWARDESS, PILOTS ARE WITHIN RANGE*

    void changePlane(Plane\* pPlane);

*// 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 POINTER TO FLIGHT, IF PASSENGER IS NOT ON THE FLIGHT ADD FLIGHT POINTER TO PASSENGER*

    void addPassenger(Passenger\* pPassenger);

*// REMOVE PASSENGER POINTER FROM FLIGHT, IF PASSENGER IS ON THE FLIGHT REMOVE FLIGHT POINTER FROM PASSENGER*

    bool removePassenger(Passenger\* pPassenger);

*// ADD STEWARDESS POINTER TO FLIGHT, IF STEWARDESS IS NOT ON THE FLIGHT ADD FLIGHT POINTER TO STEWARDESS*

    void addStewardess(CrewMember\* pStewardess);

*// REMOVE STEWARDESS POINTER FROM FLIGHT, IF STEWARDESS IS ON THE FLIGHT REMOVE FLIGHT POINTER FROM STEWARDESS*

    bool removeStewardess(CrewMember\* pStewardess);

*// ADD PILOT POINTER TO FLIGHT, IF PILOT IS NOT ON THE FLIGHT ADD FLIGHT POINTER TO PILOT*

    void addPilots(CrewMember\* pPilot);

*// REMOVE PILOT POINTER FROM FLIGHT, IF PILOT IS ON THE FLIGHT REMOVE FLIGHT POINTER FROM PILOT*

    bool removePilots(CrewMember\* pPilot);

*// REMOVE PLANE, ALL PASSENGERS, STEWARDESS, PILOTS FROM FLIGHT AND FLIGHT FROM THEM*

    void terminate();

*// CHECK IF PASSENGER, STEWARDESS, PILOT IS ON THE FLIGHT*

    bool existPassenger(Passenger\* pPassenger);

    bool existStewardess(CrewMember\* pStewardess);

    bool existPilot(CrewMember\* pPilot);

*// CHECK FOR FLIGHT TIME OVERLAP WITH TIME PERIOD*

    bool timeOverlap(const unsigned int timeStart, const unsigned int timeEnd);

   private:

*// SET PLANE CALLED IN CONSTRUCTOR WITHOUT ANY CHECK ON CAPACITY LIMITS*

    void setPlane(Plane\* pPlane);

*// 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;

    Plane\* pPlane;

    unsigned int timeDeparture; *// time in miliseconds*

    unsigned int timeArrival; *// time in miliseconds*

    std::string cityDeparture;

    std::string cityArrival;

    std::vector<Passenger\*> passengers;

    std::vector<CrewMember\*> stewardesses;

    std::vector<CrewMember\*> pilots;

};

# Testing

TEST(FlightClass, constructor) {

    Company\* company = new (Company){"Test1"};

    Flight flight1{company, "RYR120", 1, 2, "Warsaw", "Berlin"};

    auto pPlane = new (Plane){company, 101, "B737", 100, 4, 2};

    Flight flight2{company, "RYR102", pPlane, 1, 2, "Poznan", "Krakow"};

    EXPECT\_EQ(flight1.getFlightNr(), "RYR120");

    EXPECT\_EQ(flight1.getTimeDeparture(), 1);

    EXPECT\_EQ(flight1.getTimeArrival(), 2);

    EXPECT\_EQ(flight1.getCityDeparture(), "Warsaw");

    EXPECT\_EQ(flight1.getCityArrival(), "Berlin");

    EXPECT\_EQ(flight2.getPlane(), pPlane);

    delete pPlane, company;

}

TEST(FlightClass, setters) {

    Company\* company = new (Company){"Test1"};

    Flight flight{company, "RYR120", 1, 2, "Warsaw", "Berlin"};

    EXPECT\_THROW(flight.setCompany(nullptr), InvalidPointer);

    EXPECT\_THROW(flight.setFlightNr("ryr"), InvalidFlightNr);

    flight.setFlightNr("ryr1");

    EXPECT\_EQ(flight.getFlightNr(), "RYR1");

    flight.setFlightNr("rYr1201");

    EXPECT\_EQ(flight.getFlightNr(), "RYR1201");

    EXPECT\_THROW(flight.setFlightNr("rYr12012"), InvalidFlightNr);

    EXPECT\_THROW(flight.setFlightNr("RY120"), InvalidFlightNr);

    EXPECT\_THROW(flight.setFlightNr("R120"), InvalidFlightNr);

    EXPECT\_THROW(flight.changeDataDeparture(2), InvalidTime);

    EXPECT\_THROW(flight.changeDataDeparture(3), InvalidTime);

    flight.changeDataDeparture(1);

    flight.changeDataDeparture(0, "aBc");

    EXPECT\_EQ(flight.getTimeDeparture(), 0);

    EXPECT\_EQ(flight.getCityDeparture(), "aBc");

    flight.changeDataDeparture(1);

    EXPECT\_THROW(flight.changeDataArrival(1), InvalidTime);

    EXPECT\_THROW(flight.changeDataArrival(0), InvalidTime);

    flight.changeDataArrival(2);

    flight.changeDataArrival(3, "aBc");

    EXPECT\_EQ(flight.getTimeArrival(), 3);

    EXPECT\_EQ(flight.getCityArrival(), "aBc");

    EXPECT\_THROW(flight.changeDataDeparture(0, ""), InvalidName);

    EXPECT\_THROW(flight.changeDataArrival(3, ""), InvalidName);

}