МІНІСТЕРСТВО ОСВІТИ І НАУКИ, МОЛОДІ ТА СПОРТУ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»
ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ
Кафедра інформаційної безпеки

Лабораторна робота №4

з дисципліни

«System Engineering»

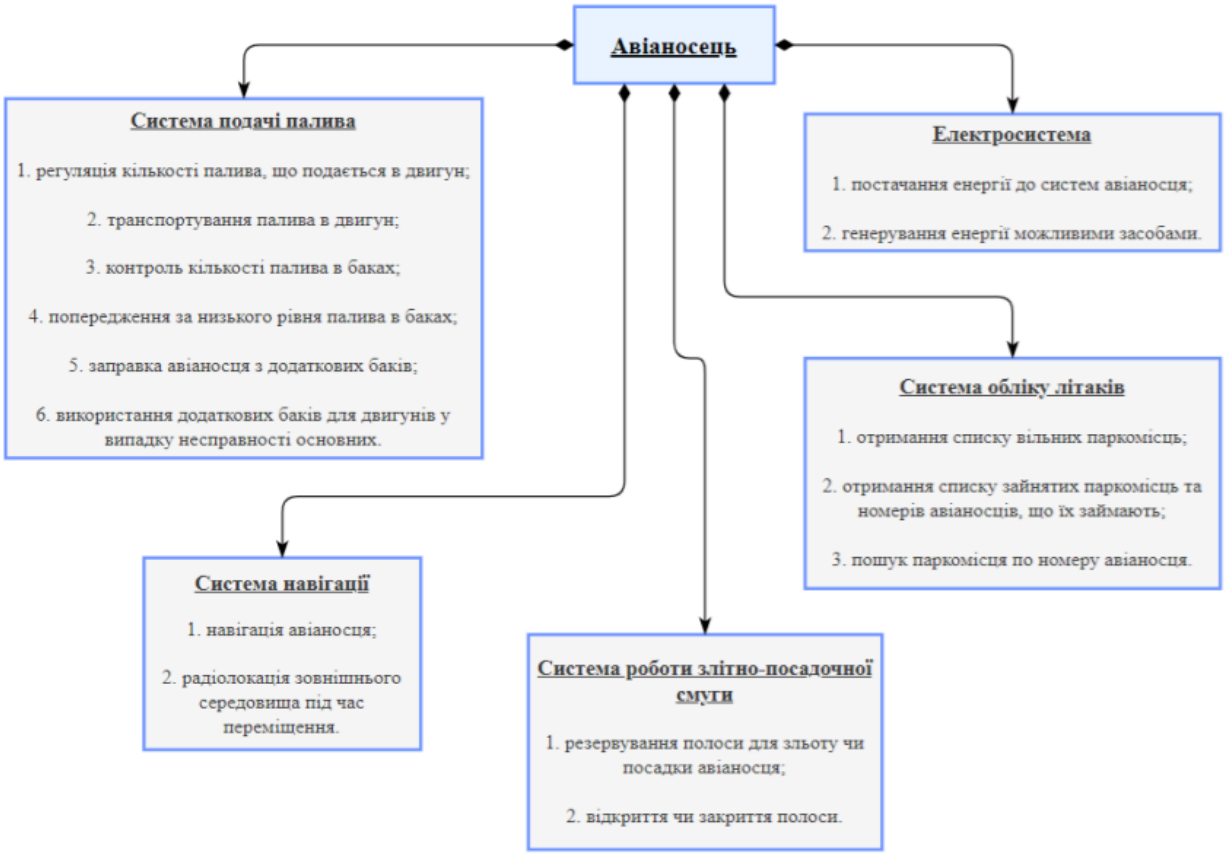
Проєкт “Авіаносець”

Перевірив:
Ткач В.М.

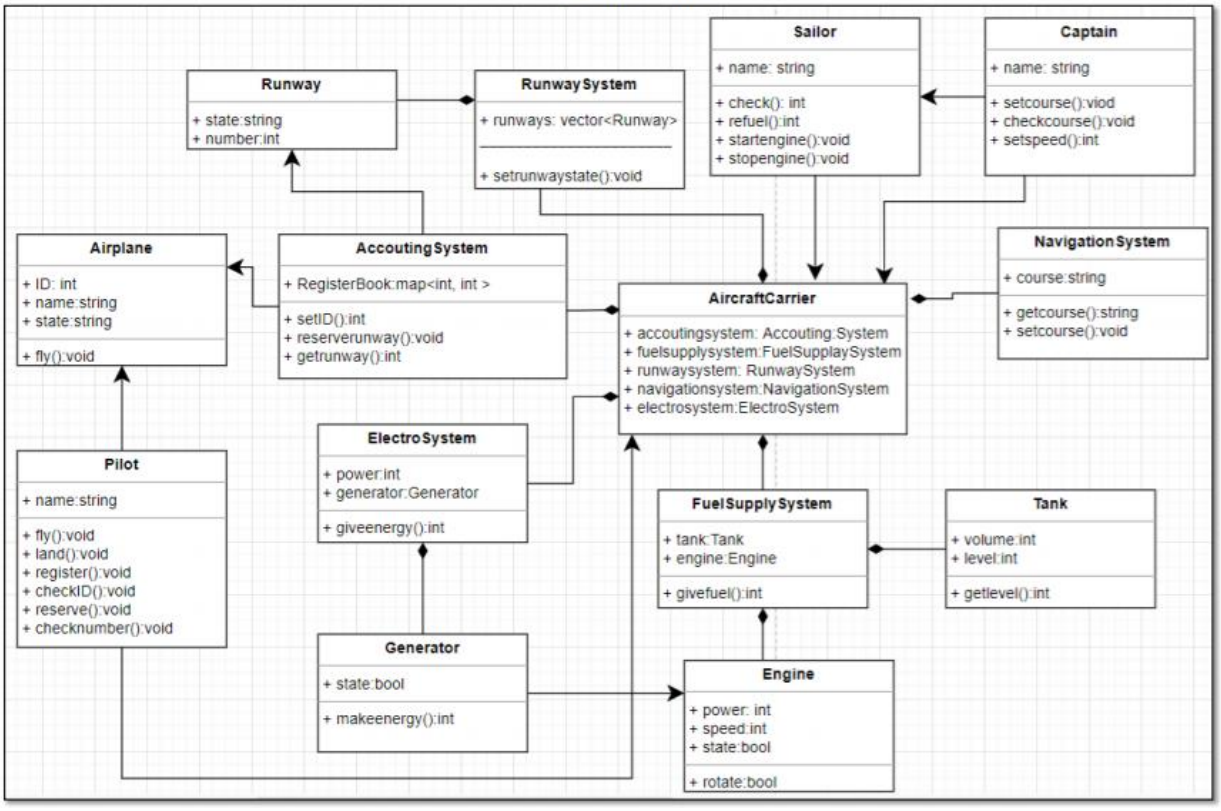
Виконав:
Студент групи ФБ-84
Киричук Т.М.

Київ 2021

Structure diagram:



Class diagram:



Завдання:

Основою проекту є діаграми, що описують систему, які були зроблені в ході виконання лабораторних робіт No1-3. Використовуючи даний матеріал необхідно максимально детально описати подану систему на будь-якій мові програмування, що підтримує об’єктно-орієнтований підхід.

Обрана мова: C++

Пояснення щодо коду:

Розглянемо використані класи та підсистеми.

Система авіаносця:

```
class AircraftCarrier {
    public:
        // деструктор
        ~AircraftCarrier(){
            delete [] planes;
        }

        // літаки, що підпорядковані авіаносцю
        Airplane planes[20];

        // екземпляри кожної із підсистем
        AccountingSystem accountingsystem;
        FuelSupplySystem fuelsupplysystem;
        RunwaySystem runwaysystem;
        NavigationSystem navigationsystem;
        ElectroSystem electrosystem;
};
```

Розглянемо кожну із підсистем, що згадані вище:

1. Система подачі палива
   Саму систему описує класс FuelSupplySystem.

```
class FuelSupplySystem {
    public:
        bool givingstate;               // стан подачі палива
        Tank tank;
        Engine engine;
        bool givefuel();                // функція подачі палива
};

bool FuelSupplySystem::givefuel() {
// якщо бак заповнений на 100%, немає необхідності заповнювати більше
    if (tank.getlevel() == 100) {
        cout << "It is already full\n";
```

```
        givingstate = false;
    } else {
        cout << "Supplying with fuel\n";
        givingstate = true;
    }
    return givingstate;
}
```

До допоміжних класів можна віднести Tank (паливний бак) та Engine (двигун авіаносця).

```
class Tank {
    int level;
    int volume;
    public:

        Tank() { level = 7000;
                 volume = 10000; };
        int getvolume();
        int getlevel();
        void setlevel(int i);
};


int Tank::getvolume() { return volume; }

int Tank::getlevel() { return level; }

void Tank::setlevel(int i) { level = i; }


class Engine {
    bool state;
    public:
        Engine() { power = 7000;
                   speed = 0;
                   state = false;
                   fuelspeed = 0; }
        int power;
        int speed;
        int fuelspeed;

        bool checkstate() { return state; };
        bool rotate();
};
bool Engine::rotate() {
    if (state) state = false;
    else state = true;
    return state;
}
```

2. Система навігації:

```
class NavigationSystem {
    string course;                                  // курс
    public:
        NavigationSystem() { course = "None"; }
        string getcourse() { return course; };
        void setcourse();                           // задання нового курсу
};

void NavigationSystem::setcourse() {
    cout << "Please enter the new destination:\n";
    cin >> course;
    cout << " -> New destination is: " << course << endl;
}
```

3. Електросистема:

```
class ElectroSystem {
    public:
        ElectroSystem() { power= 7000; };
        int power;
        Generator generator;
        void giveenergy();
};


void ElectroSystem::giveenergy() {
      if (generator.makeenergy()) cout << "Generator is working\n";
}
```

Тут допоміжним підкласом слугує генератор

```
class Generator {
    public:
        Generator() { state = false; }              // за замовчуванням
                                                       генератор вимкнений
        bool state;                                 // стан генератора


        bool makeenergy();
};

bool Generator::makeenergy(){
   state = true;
   return state;
}
```

4. Система обліку літаків:

```
class AccountingSystem {
    public:
```

```
        map <int, int> RegisterBook;            // книга регестрації
        int setID(Airplane& airplane);          // видання ID
        bool getrunaway(int plane);
};


bool AccountingSystem::getrunaway(int plane) {
      // -1 = N\A
      if( RegisterBook[plane] == -1) return true;
      else return false;
}
```

5. Система роботи злітно-посадочної смуги:

```
class RunwaySystem {
    public:
        Runway runways[20];
        void preparing_runways();
        void setrunwaystate(int index, int state);
};

// створення розмітки – кожна злітно-посадочна смуга має порядковий номер
void RunwaySystem::preparing_runways() {
      for (int i = 0; i < 20; i++ ){
            runways[i].number = i;
      }
}


// задання стану смуги
void RunwaySystem::setrunwaystate(int index, int state) {
      runways[index].state = runways[index].states_list[state];
}
```

Додатковий клас – злітно-посадочна смуга:

```
class Runway {
   public:
      // за замовчцванням закрита
      Runway() { state = states_list[0]; }
      // всі можливі стани
      string states_list[4] = {"closed", "free", "takeoff", "landing"};
      string state;
      int number;
};
```

Для взаємодії та керування підсистемами були створені також класи акторів, а саме: капітан, матрос та пілот.

1. Капітан

```
class Captain {
    string name ;
    public:
        Captain(string data) { name = data; }
        void setcourse(NavigationSystem& ns);       // встановлення курсу корабля
        void checkcourse(NavigationSystem& ns);     // перевірка курсу
        void setspeed(Engine& engine);          // встановлення швидкості корабля
        int setfuelspeed();                // встановлення швидкості подачі палива
};


void Captain::setcourse(NavigationSystem& ns) {
    ns.setcourse();                         // взаємодія із навігаційною системою
}

void Captain::checkcourse(NavigationSystem& ns) {
    // взаємодія із навігаційною системою
    cout << " -> Current destination: " << ns.getcourse() << endl;
}

void Captain::setspeed(Engine& engine) {
// встановити нову швидкість можна лише якщо двигуни працюють
if (engine.checkstate()) {
        cout << "Enter the speed: ";
        cin >> engine.speed;
    }
    cout << " -> Engine speed: " << engine.speed << endl;
    // вимкнення двигунів, якщо була встановлена швидкість 0
    if (engine.speed == 0 && engine.checkstate()) engine.rotate(); }
```

2. Матрос

```
class Sailor {
    string name;
    public:
        Sailor(string data) { name = data; }
        float check(FuelSupplySystem& fss);          // перевірка к-сті палива
        void refuel(FuelSupplySystem& fss);          // заправити
        void startengine(AircraftCarrier& aircraft); // запуск двигунів
        void stopengine(AircraftCarrier& aircraft);  // вимкнення двигунів вручну
        void action(AircraftCarrier& aircraft);      // виконання додаткових дій
        // змінити швидкість подачі палива
        void setfuelspeed(Captain& captain, FuelSupplySystem& fss);
};
```

```
float Sailor::check(FuelSupplySystem& fss) {
    int volume = fss.tank.getvolume()/100;  // для більш чітких розрахунків
    int level = fss.tank.getlevel()/100;
    float perc = (100/volume)*level;        // розрахування відсотків
    if (perc < 50) refuel(fss);      // заправити, якщо залишилось менше половини
    return perc;
}

void Sailor::refuel(FuelSupplySystem& fss) {
    cout << "Refuelling\n";
    fss.tank.setlevel(fss.tank.getvolume());
    cout << " -> Current state: " << check(fss) << endl;
}

void Sailor::setfuelspeed (Captain& captain, FuelSupplySystem& fss) {
    fss.engine.fuelspeed = captain.setfuelspeed();
    cout << " -> Current fuel speed: " << fss.engine.fuelspeed << endl;
    if (fss.engine.fuelspeed == 0) {
        cout << "Engines were stopped\n";
    }
}
```

3. Пілот

```
class Pilot {
    string name;
    public:
        Pilot(string data) { name = data; }
        // керування польоту
        void fly(Airplane& plane, RunwaySystem& rs, int i);
        // приземлення
        void land(Airplane& plane, RunwaySystem& rs, int i);
        // регестрація нового літака
        int regist(AircraftCarrier& aircraft, int i);
        // перевірка наявності ID у літака
        int checkID(AircraftCarrier& aircraft, int i);
        // резервування смуги
        int reserve(AircraftCarrier& aircraft, int plane, int i);
        // перевірка наявності резервації смуги
        int checknumber(AircraftCarrier& aircraft, int plane, int i, int m);
};


int Pilot::checkID(AircraftCarrier& aircraft, int i) {
    int answ;
    char ans;

    cout << "Which plane are you looking for?\nPlease enter the ID\n";
    cin >> answ;
    // регестація розпочинається у випадку, коли не було знайдено потрібного ID у
 обліковій книзі
```

```
    if ( aircraft.accountingsystem.RegisterBook.find(answ) == aircraft.accounting
system.RegisterBook.end()) {
        cout << "There are no plane with this ID\n";
        return regist(aircraft, i);
    }
}

int Pilot::regist(AircraftCarrier& aircraft, int i) {
    char an;

    cout << "Do you want to register it?\n";
    cin >> an;
    if (an == 'y' || an == 'Y') {
        return aircraft.accountingsystem.setID(aircraft.planes[i]);
    }
}

int Pilot::checknumber(AircraftCarrier& aircraft, int plane, int i, int m) {
    cout << "Searching for the reserved runway...\n";
    // резервація розпочинається у випадку, якщо не було знайдено відповідного за
пису в обліковій книзі
    if (aircraft.accountingsystem.getrunaway(plane)) {
        cout << "This plane doesn`t have a reserved runway\n";
        cout << "Starting the registration\n";
        cout << " -> Reserved runway: " << reserve(aircraft, plane, m) << endl;
    } else {
        // якщо ж знайдено - продовжуємо роботу із даними
        aircraft.runwaysystem.setrunwaystate(i, 1);
        cout << " -
> Reserved runway: " << aircraft.accountingsystem.RegisterBook[plane] << endl;
    }
    aircraft.runwaysystem.setrunwaystate(i, 1);
    return aircraft.accountingsystem.RegisterBook[plane];
}

int Pilot::reserve(AircraftCarrier& aircraft, int plane, int i) {
    // обмеження - 20 смуг
    if (i < 20) {
        aircraft.accountingsystem.RegisterBook[plane] = i;
        aircraft.runwaysystem.setrunwaystate(i, 1);
        return i;
    } else cout << "There are no available parking places\n";
}

void Pilot::land(Airplane& plane, RunwaySystem& rs, int i) {
    rs.setrunwaystate(i, 1);
    // початок лише у випадку якщо смуга має стан "відкрито"
    if (rs.runways[i].state == rs.runways[i].states_list[1]) {
        rs.setrunwaystate(i, 3);
        cout << " -> Current runway state: " << rs.runways[i].state << endl;
        plane.state = "landing";
```

```
        cout << " -> Plane state: " << plane.state << endl;
        plane.state = "landed";
        cout << " -> Plane state: " << plane.state << endl;
        rs.setrunwaystate(i, 0);
    } else cout << "Runway is not ready\n";
}

void Pilot::fly(Airplane& plane, RunwaySystem& rs, int i) {
    // після приземлення стан смуги був змінений на "закрито", тому для взлету не
обхідно повернутися до стану "відкрито"
    if (rs.runways[i].state == rs.runways[i].states_list[0]) {
        rs.setrunwaystate(i, 1);
        cout << " -> Current runway state: " << rs.runways[i].state << endl;
    } else cout << "Runway is not ready\n";
    // політ лише у випадку якщо смуга відкрита
    if (rs.runways[i].state == rs.runways[i].states_list[1]) {
        plane.fly();
        rs.setrunwaystate(i, 0);
        cout << " -> Current runway state: " << rs.runways[i].state << endl;
    }
}
```

Допоміжний клас – літак:

```
class Airplane {
    public:
        // за замовчуванням - очікування
        Airplane() { state = "waiting";}
        int ID;
        string name;                // назва літака
        string state;               // стан
        void fly();                 // політ
};


void Airplane::fly() {
    state = "ready to take off";
    cout << " -> Plane state: " << state << endl;
    state = "flying";
    cout << " -> Plane state: " << state << endl;
}
```

Основний процес та використання класів відбувається у функції main():

```
int main(){
    Sailor sailor("Anna Kyōyama");
    Captain captain("Yo Asakura");
    AircraftCarrier aircraft0;
    Pilot pilot01("Ren Tao");
```

```
    char ans;
    int planeID;
    int runwayID;

    // aircraft
    // sailor works
    cout << "Preparing an aircraft\n";
    cout << " -
> Current state of fuel: " << sailor.check(aircraft0.fuelsupplysystem) << endl;
    sailor.startengine(aircraft0);

    // captain works
    captain.checkcourse(aircraft0.navigationsystem);
    captain.setcourse(aircraft0.navigationsystem);
    captain.setspeed(aircraft0.fuelsupplysystem.engine);
    sailor.setfuelspeed(captain, aircraft0.fuelsupplysystem);

    int i = 0;
    int m = 0;
    do {
        planeID = pilot01.checkID(aircraft0, i);
        runwayID = pilot01.checknumber(aircraft0, planeID, i, m++);
        pilot01.land(aircraft0.planes[i], aircraft0.runwaysystem, aircraft0.accou
ntingsystem.RegisterBook[runwayID]);
        pilot01.fly(aircraft0.planes[i], aircraft0.runwaysystem, i);
        //i++;
        cout << "Are you interested in other planes?\n";
        cin >> ans;
    } while (ans == 'y' || ans == 'Y');

    sailor.action(aircraft0);

    return 0;
}
```

Приклад виконання:

```
Preparing an aircraft
 -> Current state of fuel: 70
Starting engines
Supplying with fuel
Engine is working
Generator is working
 -> Current destination: None
Please enter the new destination:
Paris
 -> New destination is: Paris
Enter the speed: 1200
 -> Engine speed: 1200
Please set fuel speed
200
 -> Current fuel speed: 200
Which plane are you looking for?
Please enter the ID
123456
There are no plane with this ID
Do you want to register it?
y
Starting the registration
Please enter the name of the plane:
Amidamaru
 -> Current information about the plane:
   1. ID: 118467
   2. Name: Amidamaru
Searching for the reserved runway...
This plane doesn`t have a reserved runway
Starting the registration
 -> Reserved runway: 0
 -> Current runway state: landing
 -> Plane state: landing
 -> Plane state: landed
 -> Current runway state: free
 -> Plane state: ready to take off
 -> Plane state: flying
 -> Current runway state: closed
Are you interested in other planes?
y
Which plane are you looking for?
Please enter the ID
132546
There are no plane with this ID
Do you want to register it?
y
Starting the registration
Please enter the name of the plane:
MarsBars
```

```
 -> Current information about the plane:
   1. ID: 106334
   2. Name: MarsBars
Searching for the reserved runway...
This plane doesn`t have a reserved runway
Starting the registration
 -> Reserved runway: 1
 -> Current runway state: landing
 -> Plane state: landing
 -> Plane state: landed
 -> Current runway state: free
 -> Plane state: ready to take off
 -> Plane state: flying
 -> Current runway state: closed
Are you interested in other planes?
n
What do you want to do?
1. Check and fix fuel supply system
2. Stop aircraft
3. Exit
1
Refuelling
 -> Current state: 100
 -> Current fuel state: 10
What do you want to do?
1. Check and fix fuel supply system
2. Stop aircraft
3. Exit
1
 -> Current fuel state: 100
What do you want to do?
1. Check and fix fuel supply system
2. Stop aircraft
3. Exit
2
Stopping the aircraft
Engines were stopped
Generator was stopped
What do you want to do?
1. Check and fix fuel supply system
2. Stop aircraft
3. Exit
2
It is already stopped
What do you want to do?
1. Check and fix fuel supply system
2. Stop aircraft
3. Exit
3
It`s all for today
```

Повний код програми:

```
#include <iostream>
#include <string>
#include <vector>
#include <map>

using namespace std;

class FuelSupplySystem;
class AircraftCarrier;
class NavigationSystem;
class Engine;
class Airplane;
class Captain;

class Tank {
    int level;
    int volume;
    public:
        Tank() { level = 7000;
                 volume = 10000; };
        int getvolume();
        int getlevel();
        void setlevel(int i);
};

class Runway {
    public:
        Runway() { state = states_list[0]; }
        string states_list[4] = {"closed", "free", "takeoff", "landing"};
        string state;
        int number;
};

class RunwaySystem {
    public:
        Runway runways[20];
        void preparing_runways();
        void setrunwaystate(int index, int state);
};

class Sailor {
    string name;
    public:
        Sailor(string data) { name = data; }
        float check(FuelSupplySystem& fss);             // перевірка к-сті палива
        void refuel(FuelSupplySystem& fss);             // заправити
        void startengine(AircraftCarrier& aircraft);    // запуск двигунів
        void stopengine(AircraftCarrier& aircraft);     // вимкнення двигунів вру
чну
```

```
        void action(AircraftCarrier& aircraft);         // виконання додаткових ді
й
        // змінити швидкість подачі палива
        void setfuelspeed(Captain& captain, FuelSupplySystem& fss);
};

class Captain {
    string name ;
    public:
        Captain(string data) { name = data; }
        void setcourse(NavigationSystem& ns);           // встановлення курсу кор
абля
        void checkcourse(NavigationSystem& ns);         // перевірка курсу
        void setspeed(Engine& engine);                  // встановлення швидкості
 корабля
        int setfuelspeed();                             // встановлення швидкості
 подачі палива
};

class NavigationSystem {
    string course;
    public:
        NavigationSystem() { course = "None"; }
        string getcourse() { return course; };
        void setcourse();
};

class Engine {
    bool state;
    public:
        Engine() { power = 7000;
                   speed = 0;
                   state = false;
                   fuelspeed = 0; }
        int power;
        int speed;
        int fuelspeed;

        bool checkstate() { return state; };
        bool rotate();
};

class FuelSupplySystem {
    public:
        bool givingstate;               // стан подачі палива
        Tank tank;
        Engine engine;
        bool givefuel();                // функція подачі палива
};

class Generator {
```

```
    public:
        Generator() { state = false; }
        bool state;                     // стан генератора
        bool makeenergy();
};

class ElectroSystem {
    public:
        ElectroSystem() { power= 7000; };
        int power;
        Generator generator;
        void giveenergy();
};

class Pilot {
    string name;
    public:
        Pilot(string data) { name = data; }
        // керування польоту
        void fly(Airplane& plane, RunwaySystem& rs, int i);
        // приземлення
        void land(Airplane& plane, RunwaySystem& rs, int i);
        // регестрація нового літака
        int regist(AircraftCarrier& aircraft, int i);
        // перевірка наявності ID у літака
        int checkID(AircraftCarrier& aircraft, int i);
        // резервування смуги
        int reserve(AircraftCarrier& aircraft, int plane, int i);
        // перевірка наявності резервації смуги
        int checknumber(AircraftCarrier& aircraft, int plane, int i, int m);
};

class Airplane {
    public:
        // за замовчуванням - очікування
        Airplane() { state = "waiting";}
        int ID;
        string name;                // назва літака
        string state;               // стан
        void fly();                 // політ
};

class AccountingSystem {
    public:
        map <int, int> RegisterBook;            // книга регестрації
        int setID(Airplane& airplane);          // видання ID
        bool getrunaway(int plane);
};

class AircraftCarrier {
    public:
```

```
        // літаки, що підпорядковані авіаносцю
        Airplane planes[20];

        // екземпляри кожної із підсистем
        AccountingSystem accountingsystem;
        FuelSupplySystem fuelsupplysystem;
        RunwaySystem runwaysystem;
        NavigationSystem navigationsystem;
        ElectroSystem electrosystem;
};

int random(int min, int max) {
    int num = min + rand() % (max - min + 1);
    return num;
}

int Tank::getlevel() { return level; }

void Tank::setlevel(int i) { level = i; }

int Tank::getvolume() { return volume; }

void RunwaySystem::preparing_runways() {
    for (int i = 0; i < 20; i++ ){
        runways[i].number = i;
    }
}

void RunwaySystem::setrunwaystate(int index, int state) {
    runways[index].state = runways[index].states_list[state];
}

float Sailor::check(FuelSupplySystem& fss) {
    int volume = fss.tank.getvolume()/100;  // для більш чітких розрахунків
    int level = fss.tank.getlevel()/100;
    float perc = (100/volume)*level;        // розрахування відсотків
    if (perc < 50) refuel(fss);             // заправити, якщо залишилось менше п
оловини
    return perc;
}

void Sailor::refuel(FuelSupplySystem& fss) {
    cout << "Refuelling\n";
    fss.tank.setlevel(fss.tank.getvolume());
    cout << " -> Current state: " << check(fss) << endl;
}

void Sailor::setfuelspeed (Captain& captain, FuelSupplySystem& fss) {
    fss.engine.fuelspeed = captain.setfuelspeed();
    cout << " -> Current fuel speed: " << fss.engine.fuelspeed << endl;
    if (fss.engine.fuelspeed == 0) {
```

```
        cout << "Engines were stopped\n";
    }
}

int Captain::setfuelspeed() {
    int speed;
    cout << "Please set fuel speed\n";
    cin >> speed;
    return speed;
}

void Captain::setcourse(NavigationSystem& ns) {
    ns.setcourse();         // взаємодія із навігаційною системою
}

void Captain::checkcourse(NavigationSystem& ns) {
    // взаємодія із навігаційною системою
    cout << " -> Current destination: " << ns.getcourse() << endl;
}

void Captain::setspeed(Engine& engine) {
    if (engine.checkstate()) {      // встановити нову швидкість можна лише якщо
двигуни працюють
        cout << "Enter the speed: ";
        cin >> engine.speed;
    }
    cout << " -> Engine speed: " << engine.speed << endl;
    if (engine.speed == 0 && engine.checkstate()) engine.rotate(); // вимкнення д
вигунів, якщо була встановлена швидкість 0
}

bool Engine::rotate() {
    if (state) state = false;
    else state = true;
    return state;
}

void NavigationSystem::setcourse() {
    cout << "Please enter the new destination:\n";
    cin >> course;
    cout << " -> New destination is: " << course << endl;
}

bool FuelSupplySystem::givefuel() {
    if (tank.getlevel() == 100) {
        cout << "It is already full\n";
        givingstate = false;
    } else {
        cout << "Supplying with fuel\n";
        givingstate = true;
    }
```

```
    return givingstate;
}

void Sailor::startengine(AircraftCarrier& aircraft) {
    cout << "Starting engines\n";
    if (aircraft.fuelsupplysystem.givefuel()) {
        if (aircraft.fuelsupplysystem.engine.rotate()) cout << "Engine is working
\n";
    }
    aircraft.fuelsupplysystem.tank.setlevel(random(1000, 9000));
    aircraft.electrosystem.giveenergy();
}

void Sailor::stopengine(AircraftCarrier& aircraft) {
    cout << "Stopping the aircraft\n";
    if (!aircraft.fuelsupplysystem.engine.rotate()) cout << "Engines were stopped
\n";
    if (aircraft.electrosystem.generator.state) {
        cout << "Generator was stopped\n";
        aircraft.electrosystem.generator.state = false;
    } else cout << "It is already stopped\n";
}

void Sailor::action(AircraftCarrier& aircraft) {
    int answ;

    do{
        cout << "What do you want to do?\n1. Check and fix fuel supply system\n2.
 Stop aircraft\n3. Exit\n";
        cin >> answ;
        switch(answ) {
            case 1: {
                cout << " -
> Current fuel state: " << check(aircraft.fuelsupplysystem) << endl;
                break; }
            case 2: {
                if (aircraft.fuelsupplysystem.engine.checkstate()) stopengine(air
craft);
                else cout << "It is already stopped\n";
                break; }
            case 3: {
                if (aircraft.fuelsupplysystem.engine.checkstate()) stopengine(air
craft);
                cout << "It`s all for today\n";
                break; }
            default: cout << "Wrong symbol\n";
        }
    } while (answ == 1 || answ == 2);
}

bool Generator::makeenergy(){
```

```
    state = true;
    return state;
}

void ElectroSystem::giveenergy() {
    if (generator.makeenergy()) cout << "Generator is working\n";
}

int Pilot::checkID(AircraftCarrier& aircraft, int i) {
    int answ;
    char ans;

    cout << "Which plane are you looking for?\nPlease enter the ID\n";
    cin >> answ;
    // регестація розпочинається у випадку, коли не було знайдено потрібного ID у
 обліковій книзі
    if ( aircraft.accountingsystem.RegisterBook.find(answ) == aircraft.accounting
system.RegisterBook.end()) {
        cout << "There are no plane with this ID\n";
        return regist(aircraft, i);
    }
}

int Pilot::regist(AircraftCarrier& aircraft, int i) {
    char an;

    cout << "Do you want to register it?\n";
    cin >> an;
    if (an == 'y' || an == 'Y') {
        return aircraft.accountingsystem.setID(aircraft.planes[i]);
    }
}

int Pilot::checknumber(AircraftCarrier& aircraft, int plane, int i, int m) {
    cout << "Searching for the reserved runway...\n";
    // резервація розпочинається у випадку, якщо не було знайдено відповідного за
пису в обліковій книзі
    if (aircraft.accountingsystem.getrunaway(plane)) {
        cout << "This plane doesn`t have a reserved runway\n";
        cout << "Starting the registration\n";
        cout << " -> Reserved runway: " << reserve(aircraft, plane, m) << endl;
    } else {
        // якщо ж знайдено - продовжуємо роботу із даними
        aircraft.runwaysystem.setrunwaystate(i, 1);
        cout << " -
> Reserved runway: " << aircraft.accountingsystem.RegisterBook[plane] << endl;
    }
    aircraft.runwaysystem.setrunwaystate(i, 1);
    return aircraft.accountingsystem.RegisterBook[plane];
}
```

```
int Pilot::reserve(AircraftCarrier& aircraft, int plane, int i) {
    // обмеження у 20 смуг
    if (i < 20) {
        aircraft.accountingsystem.RegisterBook[plane] = i;
        aircraft.runwaysystem.setrunwaystate(i, 1);
        return i;
    } else cout << "There are no available parking places\n";
}

void Pilot::land(Airplane& plane, RunwaySystem& rs, int i) {
    rs.setrunwaystate(i, 1);
    // початок лише у випадку якщо смуга має стан "відкрито"
    if (rs.runways[i].state == rs.runways[i].states_list[1]) {
        rs.setrunwaystate(i, 3);
        cout << " -> Current runway state: " << rs.runways[i].state << endl;
        plane.state = "landing";
        cout << " -> Plane state: " << plane.state << endl;
        plane.state = "landed";
        cout << " -> Plane state: " << plane.state << endl;
        rs.setrunwaystate(i, 0);
    } else cout << "Runway is not ready\n";
}

void Pilot::fly(Airplane& plane, RunwaySystem& rs, int i) {
    // після приземлення стан смуги був змінений на "закрито", тому для взлету не
обхідно повернутися до стану "відкрито"
    if (rs.runways[i].state == rs.runways[i].states_list[0]) {
        rs.setrunwaystate(i, 1);
        cout << " -> Current runway state: " << rs.runways[i].state << endl;
    } else cout << "Runway is not ready\n";
    // політ ише у випадку якщо смуга відкрита
    if (rs.runways[i].state == rs.runways[i].states_list[1]) {
        plane.fly();
        rs.setrunwaystate(i, 0);
        cout << " -> Current runway state: " << rs.runways[i].state << endl;
    }
}

void Airplane::fly() {
    state = "ready to take off";
    cout << " -> Plane state: " << state << endl;
    state = "flying";
    cout << " -> Plane state: " << state << endl;
}

int AccountingSystem::setID(Airplane& airplane){
    int ID;

    cout << "Starting the registration\n";
    do {
        // генерація 6-значного числа згідно вимог до ID
```

```
        ID = random(100000,999999);
        // якщо у книзі не знайдено потрібних данних, додаємо нові
        if (RegisterBook.find(ID) == RegisterBook.end()) {
            airplane.ID = ID;
            // додання у книгу
            RegisterBook.insert(pair<int, int>(ID, -1));
        }
    } while (RegisterBook.find(ID) == RegisterBook.end());

    cout << "Please enter the name of the plane: \n";
    cin >> airplane.name;
    cout << " -
> Current information about the plane: \n" << "   1. ID: " << airplane.ID << endl
;
    cout << "   2. Name: " << airplane.name << endl;
    return ID;
}

bool AccountingSystem::getrunaway(int plane) {
   if( RegisterBook[plane] == -1) return true;
   else return false;
}

int main(){
    Sailor sailor("Anna Kyōyama");
    Captain captain("Yo Asakura");
    AircraftCarrier aircraft0;
    Pilot pilot01("Ren Tao");
    char ans;
    int planeID;
    int runwayID;

    // aircraft
    // sailor works
    cout << "Preparing an aircraft\n";
    cout << " -
> Current state of fuel: " << sailor.check(aircraft0.fuelsupplysystem) << endl;
    sailor.startengine(aircraft0);

    // captain works
    captain.checkcourse(aircraft0.navigationsystem);
    captain.setcourse(aircraft0.navigationsystem);
    captain.setspeed(aircraft0.fuelsupplysystem.engine);
    sailor.setfuelspeed(captain, aircraft0.fuelsupplysystem);

    int i = 0;
    int m = 0;
    do {
        planeID = pilot01.checkID(aircraft0, i);
        runwayID = pilot01.checknumber(aircraft0, planeID, i, m++);
```

```
        pilot01.land(aircraft0.planes[i], aircraft0.runwaysystem, aircraft0.accou
ntingsystem.RegisterBook[runwayID]);
        pilot01.fly(aircraft0.planes[i], aircraft0.runwaysystem, i);
        //i++;
        cout << "Are you interested in other planes?\n";
        cin >> ans;
    } while (ans == 'y' || ans == 'Y');

    sailor.action(aircraft0);

    return 0;
}
```