**CFlyObject.h**

```
#pragma once
#include<iostream>
using namespace std;
//0 nu afisez nimic, 1 afisez apeluri constr etc
static int flag = 0;
enum FlyObjectType {
    AIR_PLANE = 1,
    GLIDER = 2
};

class CFlyObject {
private:
    static int m_counter;
protected:
    char* m_ownerName;
    int m_speed;
    double m_load;
    CFlyObject();
    CFlyObject(const char* nume, int speed = 0, double load = 0);
    CFlyObject(const CFlyObject &obj);
public:
    ~CFlyObject();
    virtual void Write();
    virtual void Read();
    static int GetContor();
    friend int CompareBySpeed(const CFlyObject *obj1, const CFlyObject *obj2);
};
```

**CAirPlane.h**

```
#pragma once
#include"CFlyObject.h"

enum PropulsionType {
   NONE = 0,
   JET = 1,
   PROPELLER = 2
};

class CAirPlane : public CFlyObject {
private:
   PropulsionType m_propulsion;
public:
   CAirPlane();
   CAirPlane(char* const nume, int speed = 0, double load = 0, PropulsionType propulsion =
NONE);
   CAirPlane(const CAirPlane &plane);
   ~CAirPlane();
   void Write();
   void Read();
   inline int GetPropulsion();
   CAirPlane& operator=(const CAirPlane &plane);
   CAirPlane operator+(double load);
   friend ostream& operator<<(ostream& os, CAirPlane &plane);
   friend istream& operator >> (istream& is, CAirPlane &plane);
};
```

**CFlyObjectFactory.h**

```
#pragma once
#include"CAirPlane.h"
#include"CGlider.h"

class CFlyObjectFactory {
public:
    static CFlyObject* CreateFlyObject(FlyObjectType type);
};
```

**CGlider.h**

```
#pragma once
#pragma once
#include"CFlyObject.h"

enum GliderClass {
   UNKNOWN = 0,
   CLUB = 1,
   STANDARD = 2,
   DUAL_SEATS = 3,
   OPEN = 4
};

class CGlider : public CFlyObject {
private:
   GliderClass m_class;
public:
   CGlider();
   CGlider(char* const nume, int speed = 0, double load = 0, GliderClass clasa = UNKNOWN);
   CGlider(const CGlider &glider);
   ~CGlider();
   void Write();
   void Read();
   inline int GetClass();
   CGlider& operator++();
   friend ostream& operator<<(ostream& os, CGlider &glider);
   friend istream& operator>> (istream& is, CGlider &glider);
   CGlider& operator=(const CGlider& glider);


};
```

**CFlyObject.cpp**

```
#include"CFlyObject.h"

int CFlyObject::m_counter = 0;

CFlyObject::CFlyObject()
{
   m_ownerName = nullptr;
   m_speed = 0;
   m_load = 0;
   ++m_counter;
   if (flag)
         cout << "Constructor CFlyObject fara argumente." << endl;
}

CFlyObject::CFlyObject(const char* nume, int speed, double load)
{
   if (nume != nullptr)
   {
         m_ownerName = new char[strlen(nume) + 1];
         strcpy_s(m_ownerName, strlen(nume) + 1, nume);
   }
   else
         m_ownerName = nullptr;
   m_speed = speed;
   m_load = load;
   ++m_counter;
   if (flag)
         cout << "Constructor CFlyObject cu argumente." << endl;
}

CFlyObject::CFlyObject(const CFlyObject &obj)
{
   if (obj.m_ownerName != nullptr)
   {
         m_ownerName = new char[strlen(obj.m_ownerName) + 1];
         strcpy_s(m_ownerName, strlen(obj.m_ownerName) + 1, obj.m_ownerName);
   }
   else
         m_ownerName = nullptr;
   m_speed = obj.m_speed;
   m_load = obj.m_load;
```

```
    ++m_counter;
    if (flag)
        cout << "Constructor CFlyObject de copiere." << endl;
}

CFlyObject::~CFlyObject()
{
    if (m_ownerName != nullptr)
        delete[] m_ownerName;
    m_ownerName = nullptr;
    --m_counter;
    if (flag)
        cout << "Destructor CFlyObject." << endl;
}

void CFlyObject::Write()
{
    cout << "\n\n Owner name: ";
    if (m_ownerName)
        cout << m_ownerName;
    cout << "\n Load: " << m_load;
    cout << "\n Speed: " << m_speed << endl;
}

void CFlyObject::Read()
{
    char tempt[100];

    cout << endl << "Introduceti viteza: ";
    cin >> m_speed;
    cout << endl << "Introduceti cantitatea de incarcatura: ";
    cin >> m_load;

    //atentie la golit continut buffer aici pt 2 citiri succesive!!!
    cout << "Introduceti numele: ";
    cin.ignore(100, '\n');
    cin.getline(tempt, 100);
    if (tempt != nullptr)
    {
        m_ownerName = new char[strlen(tempt) + 1];
        strcpy_s(m_ownerName, strlen(tempt) + 1, tempt);
    }
    else
```

```
        m_ownerName = nullptr;
}

int CFlyObject::GetContor()
{
    return m_counter;
}

int CompareBySpeed(const CFlyObject *obj1, const CFlyObject *obj2)
{
    return obj1->m_speed > obj2->m_speed;
}
```

**CAirPlane.cpp**

```
#include"CAirPlane.h"

CAirPlane::CAirPlane() : CFlyObject()
{
   m_propulsion = NONE;
   if (flag)
         cout << "Constructor fara argumente CAirPlane." << endl;
}
CAirPlane::CAirPlane(char* const nume, int speed, double load, PropulsionType propulsion) :
CFlyObject(nume, speed, load)
{
   m_propulsion = propulsion;
   if (flag)
         cout << "Constructor cu argumente CAirPlane." << endl;
}
CAirPlane::CAirPlane(const CAirPlane &plane) : CFlyObject(plane)
{
   m_propulsion = plane.m_propulsion;
   if (flag)
         cout << "Constructor de copiere CAirPlane." << endl;
}
CAirPlane::~CAirPlane()
{
   m_propulsion = NONE;
   if (flag)
         cout << "Destructor fara argumente CAirPlane." << endl;
}
void CAirPlane::Write()
{
   CFlyObject::Write();
   cout << "Tipul este: ";

   switch (m_propulsion)
   {
   case NONE:
         cout << "none.";
         break;
   case JET:
         cout << "jet.";
         break;
   case PROPELLER:
```

```
            cout << "propeller.";
            break;
    }
    cout << endl;
}
void CAirPlane::Read()
{
    CFlyObject::Read();

    unsigned int opt;
    do{

            cout << "Introduceti tipul: \n [1] - Jet. \n [2] - Propeller. \n";
            cin >> opt;
            if(opt<1 || opt>2)
                    cout << "Optiune gresita. Introduceti din nou:" << endl;
            else
                    switch (opt)
                    {
                    case 1:
                            m_propulsion = JET;
                            break;
                    case 2:
                            m_propulsion = PROPELLER;
                            break;
                    }
    } while (opt < 1 || opt>2);

}
inline int CAirPlane::GetPropulsion()
{
    return m_propulsion;
}
CAirPlane& CAirPlane::operator=(const CAirPlane &plane)
{
    if (plane.m_ownerName != nullptr)
    {
            m_ownerName = new char[strlen(plane.m_ownerName) + 1];
            strcpy_s(m_ownerName, strlen(plane.m_ownerName) + 1, plane.m_ownerName);
    }
    else
            m_ownerName = nullptr;
    m_speed = plane.m_speed;
```

```
    m_load = plane.m_load;
    m_propulsion = plane.m_propulsion;
    return *this;

}
CAirPlane CAirPlane::operator+(double load)
{
    m_load += load;
    return *this;
}
ostream& operator<<(ostream& os, CAirPlane &plane)
{
    os << "\n\n Owner name: ";
    if (plane.m_ownerName)
            os << plane.m_ownerName;
    os << "\n Speed: " << plane.m_speed << " km/h\n Load: " << plane.m_load << " kg \n
Propusion: ";
    switch (plane.m_propulsion)
    {
    case NONE:
            os << " none.";
            break;
    case JET:
            os << " jet.";
            break;
    case PROPELLER:
            os << " propeller.";
            break;
    }
    os << endl;
    return os;
}
istream& operator>>(istream& is, CAirPlane &plane)
{

    int aclass;
    char tempt[100];
          cout << "\nAirplane propulsion (0-1-2):";
    is >> aclass;
    switch (aclass)
    {
    case 0:
            plane.m_propulsion = NONE;
```

```
            break;
        case 1:
            plane.m_propulsion = JET;
            break;
        case 2:
            plane.m_propulsion = PROPELLER;
            break;
        }
        cout << "\nAirplane speed:";
        is >> plane.m_speed;
        cout << "\nAirplane load:";
        is >> plane.m_load;
        cout << "\n Owner name: ";
        cin.ignore(100, '\n');
        cin.getline(tempt, 100);
        if (tempt != nullptr)
        {
            plane.m_ownerName = new char[strlen(tempt) + 1];
            strcpy_s(plane.m_ownerName, strlen(tempt) + 1, tempt);
        }
        else
            plane.m_ownerName = nullptr;
        return is;
    }
```

**CFlyObjectFactory.cpp**

```
#include"CFlyObjectFactory.h"
CFlyObject* CFlyObjectFactory::CreateFlyObject(FlyObjectType type)
{
   CFlyObject* obj = nullptr;
   switch (type)
   {
   case AIR_PLANE:
         obj = new CAirPlane();
         break;
   case GLIDER:
         obj = new CGlider();
         break;
   }
   return obj;
}
```

**CGlider.cpp**

```
#include"CGlider.h"

CGlider::CGlider() : CFlyObject()
{
   m_class = UNKNOWN;
   if (flag)
         cout << "Constructor fara argumente CFlyObject." << endl;
}
CGlider::CGlider(char* const nume, int speed, double load, GliderClass clasa) :
CFlyObject(nume, speed, load)
{
   m_class = clasa;
   if (flag)
         cout << "Constructor cu argumente CFlyObject." << endl;
}
CGlider::CGlider(const CGlider &glider) : CFlyObject(glider)
{
   m_class = glider.m_class;
   if (flag)
         cout << "Constructor de copiere CFlyObject." << endl;
}
CGlider::~CGlider()
{
   m_class = UNKNOWN;
   if (flag)
         cout << "Destructor CFlyObject." << endl;
}
void CGlider::Write()
{
   CFlyObject::Write();
   cout << "Clasa este: ";
   switch (m_class)
   {
   case UNKNOWN:
         cout << "Unknown.";
         break;
   case CLUB:
         cout << "Club.";
         break;
   case STANDARD:
         cout << "Standard.";
```

```
            break;
    case DUAL_SEATS:
            cout << "Dual Seats.";
            break;
    case OPEN:
            cout << "Open.";
            break;
    }
    cout << endl;
}
void CGlider::Read()
{
    CFlyObject::Read();
    cout << "Introduceti tipul: \n [1] - Club. \n [2] - Standard. \n [3] - Dual Seat. \n [4] - Open. \n";
    unsigned int opt;
    cin >> opt;
    while ((opt < 1) || (opt >4))
    {
            cout << "Optiune gresita. Introduceti din nou:" << endl;
            cin >> opt;
    }
    switch (opt)
    {
    case 1:
            m_class = CLUB;
            break;
    case 2:
            m_class = STANDARD;
            break;
    case 3:
            m_class = DUAL_SEATS;
            break;
    case 4:
            m_class = OPEN;
            break;
    }
}
inline int CGlider::GetClass()
{
    return m_class;
}
CGlider& CGlider::operator++()
{
```

```
    if (flag)
        cout << "\n Glider++()\n";
    switch (this->m_class)
    {
    case UNKNOWN:
        this->m_class = CLUB;
        break;
    case CLUB:
        this->m_class = STANDARD;
        break;
    case STANDARD:
        this->m_class = DUAL_SEATS;
        break;
    case DUAL_SEATS:
        this->m_class = OPEN;
        break;
    case OPEN:
        this->m_class = UNKNOWN;
        break;
    }
    return *this;
}

CGlider& CGlider::operator=(const CGlider& glider)
{
    m_load = glider.m_load;
    m_speed = glider.m_speed;
    m_ownerName = new char[strlen(glider.m_ownerName) + 1];
    strcpy_s(m_ownerName, strlen(glider.m_ownerName) + 1, glider.m_ownerName);
    m_class = glider.m_class;
    cout << "operator= CGlider." << endl;
    return *this;
}

ostream& operator<<(ostream& os, CGlider &glider)
{
    os << "\n\nOwner name: ";
    if (glider.m_ownerName)
        os << glider.m_ownerName;
    os << "\nSpeed: " << glider.m_speed << " km/h\nLoad: " << glider.m_load << " kg\nClass: ";
    switch (glider.m_class)
    {
    case UNKNOWN:
```

```
            os << " Unknown.";
            break;
    case CLUB:
            os << " Club.";
            break;
    case STANDARD:
            os << " Standard.";
            break;
    case DUAL_SEATS:
            os << " Dual Seats.";
            break;
    case OPEN:
            os << " Open.";
            break;
    }
    os << "\n";
    return os;
}
istream& operator>>(istream& is, CGlider &glider)
{
    int gclass;
    char tempt[100];

    cout << "\Glider class (0-1-2-3-4):";
    is >> gclass;
    switch (gclass)
    {
    case 0:
            glider.m_class = UNKNOWN;
            break;
    case 1:
            glider.m_class = CLUB;
            break;
    case 2:
            glider.m_class = STANDARD;
            break;
    case 3:
            glider.m_class = DUAL_SEATS;
            break;
    case 4:
            glider.m_class = OPEN;
            break;
    }
```

```
    cout << "\nGlider speed:";
    is >> glider.m_speed;
    cout << "\nGlider load:";
    is >> glider.m_load;
    cout << "\nOwner name: ";
    cin.ignore(100, '\n');
    cin.getline(tempt, 100);
    if (tempt != nullptr)
    {
        glider.m_ownerName = new char[strlen(tempt) + 1];
        strcpy_s(glider.m_ownerName, strlen(tempt) + 1, tempt);
    }
    else
        glider.m_ownerName = nullptr;
    return is;
}
```

**Main.cpp**

```
#include "CFlyObjectFactory.h"

int main()
{
   // Pentru nota 5 din barem instanţiaţi obiecte de tipul claselor derivate
   //astfel încât să puneţi în evidenţă apelul fiecărui constructor pentru fiecare clasă în parte
   cout << "\n----------Instantiere airplane si glider----------\n\n\n";

   CAirPlane a1("Airplane1", 120, 2400, JET);
   CAirPlane a2;
   CAirPlane a3(a1);
   CAirPlane a4, a5;
   CGlider g1("Glider1", 80, 15, STANDARD);
   CGlider g2;
   CGlider g3(g2);
   CGlider g4, g5;

   a1.Write();
   a2.Write();
   a3.Write();
   g1.Write();
   g2.Write();
   g3.Write();

   //le-am lasat comentate, functioneaza, dar nu are sens sa pierdem timp cu introd date de fiec
data cand rulam!
   //le decomentati la corectura daca e cazul
   //trebuie testate oricum un pic mai jos
   //a4.Write();
   //g4.Write();
   //a4.Read();
   //g4.Read();
   //a4.Write();
   //g4.Write();

   //Numar instante
   cout << "\n----------Numar instante----------\n\n\n";
   cout << "\n Numar instante: " << CFlyObject::GetContor() << endl;

   //puneţi în evidenţă apelul fiecărei funcţii operator, inclusiv operator<< şi operator>>
   cout << "\n----------airplane operator=----------\n\n\n";
```

```
    a5.Write();
    a5 = a1;
    a5.Write();
    cout << "\n----------airplane operator+----------\n\n\n";
    a5 = a5 + 1204;
    a5.Write();
    cout << "\n----------glider operator++----------\n\n\n";
    g5.Write();
    ++g5;
    g5.Write();
    cout << "\n----------operator>>----------\n\n\n";
    //le comentati/decomentati la corectura daca e cazul
    cin >> a5;
    cin >> g5;
    cout << "\n----------operator<<----------\n\n\n";
    cout << a5;
    cout << g5;


    //în funcția main() alocați dinamic memorie pentru un vector de pointeri către clasa de bază
    cout << "\n----------vectori de cflyobjects----------\n\n\n";
    int n = 4; //n=10 daca nu faceti citire
    CFlyObject **vp = new CFlyObject*[n];

    /*
    vp[0] = new CAirPlane(a1);
    vp[1] = new CAirPlane(a2);
    vp[2] = new CAirPlane(a3);
    vp[3] = new CAirPlane(a4);
    vp[4] = new CAirPlane(a5);
    vp[5] = new CGlider(g1);
    vp[6] = new CGlider(g2);
    vp[7] = new CGlider(g3);
    vp[8] = new CGlider(g4);
    vp[9] = new CGlider(g5);
    */
    cout << endl<<"----parcurgeti vectorul si creati dinamic obiecte cu ajutorul functiei statice a
clasei Factory----"<<endl<<endl<<endl;
    cout << "\n----parcurgeti vectorul si cititi membrii obiectelor create (polimorfism)----\n\n\n";
    for (int i = 0; i < n; i+=2)
    {
            vp[i] = CFlyObjectFactory::CreateFlyObject(AIR_PLANE);
            vp[i + 1] = CFlyObjectFactory::CreateFlyObject(GLIDER);
```

```
		vp[i]->Read();
		vp[i + 1]->Read();

	}

	cout << "\n----parcurgeti vectorul si afisati valorile membrilor obiectelor (polimorfism)--\n\n\n";
	for (int i = 0;i < n;i++)
	{
		vp[i]->Write();
	}

	cout << "\n---sortati elementele vectorului folosind functia prietena CompareBySpeed(…) si
reafisati vectorul---\n\n\n";
	CFlyObject *aux;
	int ok = 1, lim = n - 1;

	do {
		ok = 0;
		for (int i = 0;i < lim;i++)
		{
			if (CompareBySpeed(vp[i], vp[i + 1])>0)
			{
				aux = vp[i];
				vp[i] = vp[i + 1];
				vp[i + 1] = aux;
				ok = 1;
			}
		}
		lim--;
	} while (ok);

	for (int i = 0;i < n;i++)
	{
		vp[i]->Write();
	}

	cout << "\n---------------------------------------eliberati
memoria---------------------------------------\n\n\n";
	for (int i = 0;i < n;i++)
	{
		if (vp[i])
			delete vp[i];
		vp[i] = nullptr;
```

```
    }
    if (vp)
            delete[] vp;
    vp = nullptr;


    system("pause");

    return 0;
}
```