|  |  |
| --- | --- |
| Институт (факультет) | Институт информационных технологий |
| Кафедра | Кафедра математического и программного обеспечения ЭВМ |

# РАЗРАБОТКА АБСТРАКТНЫХ ТИПОВ ДАННЫХ

|  |  |
| --- | --- |
| **Дисциплина:** | ООП |
| **Темы:** | **Наследование** |

**Среда разработки:** Microsoft Visual Studio

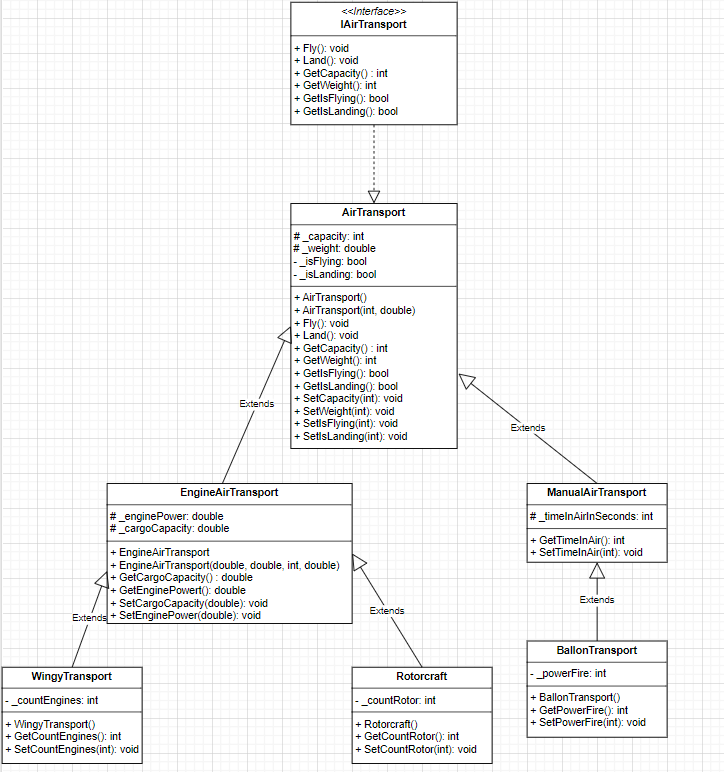
**Язык программирования:** C++

**Тип проекта:** Консольное приложение

**ТРЕБОВАНИЯ К РАЗРАБОТКЕ**

1. Каждый класс должен быть оформлен в отдельных файлах: заголовочный (.h) и файл с кодом (.cpp).
2. Запрещается использовать обработку исключительных ситуаций и генерировать исключения.
3. Придерживайтесь принципа DRY (Don’t repeat yourself).
4. Обязательно наличие комментариев.

Схема:



Код:

sem2.cpp

#include <iostream>

#include "AirTransport.h"

#include "EngineAirTransport.h"

#include "WingyTransport.h"

#include "Rotorcraft.h"

#include "BallonTransport.h"

using namespace std;

void getCapacity(IAirTransport& transport) {

cout << "Вместимость" << transport.GetCapacity() << endl;

}

void getEnginePower(EngineAirTransport tranport) {

cout << "Мощность двигателя" << tranport.GetEnginePower() << endl;

}

int main()

{

setlocale(LC\_ALL, "rus");

WingyTransport wingy;

Rotorcraft helicopter;

BallonTransport ballon;

getCapacity(wingy);

getCapacity(helicopter);

getCapacity(ballon);

getEnginePower(wingy);

getEnginePower(helicopter);

}

IAirTransport.h

#pragma once

class IAirTransport

{

public:

/// <summary>Возвращает вместимость транспорта.</summary>

/// <returns>Вместимость транспорта.</returns>

virtual int GetCapacity() = 0;

/// <summary>Возвращает вес транспорта.</summary>

/// <returns>Вес транспорта.</returns>

virtual double GetWeight() = 0;

/// <summary>Возвращает в воздухе ли транспорт.</summary>

/// <returns>Воздухе ли транспорт.</returns>

virtual bool GetIsFlying() = 0;

/// <summary>Возвращает приземлился ли транспорт.</summary>

/// <returns>Приземлился ли транспорт.</returns>

virtual bool GetIsLanding() = 0;

/// <summary>Позволяет взлететь.</summary>

virtual void Fly() = 0;

/// <summary>Приземлиться.</summary>

virtual void Land() = 0;

};

AirTransport.h

#pragma once

#include "IAirTransport.h"

class AirTransport : public IAirTransport

{

private:

bool \_isFlying;

bool \_isLanding;

protected:

int \_capacity;

double \_weight;

public:

AirTransport();

/// <summary>Создание объекта AirTransport.</summary>

/// <param name="capacity">Вместимость.</param>

/// <param name="weight">Вес.</param>

AirTransport(int capacity, double weight);

/// <summary>Возвращает вместимость транспорта.</summary>

/// <returns>Вместимость транспорта.</returns>

int GetCapacity();

/// <summary>Задает вместимость транспорта.</summary>

/// <param name="capacity">Вместимость транспорта.</param>

void SetCapacity(int capacity);

/// <summary>Возвращает вес транспорта.</summary>

/// <returns>Вес транспорта.</returns>

double GetWeight();

/// <summary>Задает вес транпорта.</summary>

/// <returns>Вес транспорта.</returns>

void SetWeight(double weight);

/// <summary>Возвращает в воздухе ли транспорт.</summary>

/// <returns>Воздухе ли транспорт.</returns>

bool GetIsFlying();

/// <summary>Возвращает приземлился ли транспорт.</summary>

/// <returns>Приземлился ли транспорт.</returns>

bool GetIsLanding();

/// <summary>Позволяет взлететь.</summary>

void Fly();

/// <summary>Приземлиться.</summary>

void Land();

};

AirTransport.cpp

#include "airTransport.h"

#include <exception>

#include "CheckValidation.h"

using namespace std;

AirTransport::AirTransport()

{

\_weight = 100;

\_capacity = 100;

\_isFlying = false;

\_isLanding = true;

}

AirTransport::AirTransport(int capacity, double weight)

{

\_capacity = CheckValidation::CheckNegative(capacity);

\_weight = CheckValidation::CheckNegative(weight);

\_isFlying = false;

\_isLanding = true;

}

int AirTransport::GetCapacity()

{

return \_capacity;

}

void AirTransport::SetCapacity(int capacity)

{

\_capacity = CheckValidation::CheckNegative(capacity);

}

double AirTransport::GetWeight()

{

return \_weight;

}

void AirTransport::SetWeight(double weight)

{

\_weight = CheckValidation::CheckNegative(weight);

}

bool AirTransport::GetIsFlying()

{

return \_isFlying;

}

bool AirTransport::GetIsLanding()

{

return \_isLanding;

}

void AirTransport::Fly()

{

\_isFlying = true;

\_isLanding = false;

}

void AirTransport::Land()

{

\_isFlying = false;

\_isLanding = true;

}

EngineAirTransport.h

#pragma once

#include "AirTransport.h"

class EngineAirTransport : public AirTransport

{

protected:

double \_enginePower;

double \_cargoCapacity;

public:

EngineAirTransport(double enginePower, double cargoCapacity, double capacity, int weight);

EngineAirTransport(double enginePower, double cargoCapacity);

EngineAirTransport();

double GetEnginePower();

double GetCargoCapacity();

void SetEnginePower(double enginePower);

void SetCargoCapacity(double cargoCapacity);

};

EngineAirTransport.cpp

#include "EngineAirTransport.h"

#include "CheckValidation.h"

EngineAirTransport::EngineAirTransport(double enginePower, double cargoCapacity, double capacity, int weight) : AirTransport(capacity, weight)

{

\_cargoCapacity = CheckValidation::CheckNegative(cargoCapacity);

\_enginePower = CheckValidation::CheckNegative(enginePower);

}

EngineAirTransport::EngineAirTransport(double enginePower, double cargoCapacity) : AirTransport()

{

\_cargoCapacity = CheckValidation::CheckNegative(cargoCapacity);

\_enginePower = CheckValidation::CheckNegative(enginePower);

}

EngineAirTransport::EngineAirTransport() : AirTransport()

{

\_cargoCapacity = 20;

\_enginePower = 200;

}

double EngineAirTransport::GetEnginePower()

{

return \_enginePower;

}

double EngineAirTransport::GetCargoCapacity()

{

return \_cargoCapacity;

}

void EngineAirTransport::SetEnginePower(double enginePower)

{

\_enginePower = CheckValidation::CheckNegative(enginePower);

}

void EngineAirTransport::SetCargoCapacity(double cargoCapacity)

{

\_cargoCapacity = CheckValidation::CheckNegative(cargoCapacity);

}

BallonTransport.h

#pragma once

#include "ManualAirTransport.h"

class BallonTransport : public ManualAirTransport

{

private:

int \_powerFire;

public:

BallonTransport();

int GetPowerFire();

void SetPowerFire(int powerFire);

};

BallonTransport.cpp

#include "BallonTransport.h"

BallonTransport::BallonTransport() : ManualAirTransport()

{

\_powerFire = 45;

}

int BallonTransport::GetPowerFire()

{

return \_powerFire;

}

void BallonTransport::SetPowerFire(int powerFire)

{

\_powerFire = powerFire;

}

WingyTransport.h

#pragma once

#include "EngineAirTransport.h"

class WingyTransport : public EngineAirTransport

{

private:

int \_countEngines;

public:

WingyTransport();

int GetCountEngines();

void SetCountEngines(int countEngines);

};

WingyTransport.cpp

#include "WingyTransport.h"

#include "CheckValidation.h"

WingyTransport::WingyTransport() : EngineAirTransport()

{

\_countEngines = 4;

}

int WingyTransport::GetCountEngines()

{

return \_countEngines;

}

void WingyTransport::SetCountEngines(int countEngines)

{

\_countEngines = CheckValidation::CheckNegative(countEngines);

}

Rotorcraft.h

#pragma once

#include "EngineAirTransport.h"

class Rotorcraft : public EngineAirTransport

{

private:

int \_countRotor;

public:

Rotorcraft();

int GetCountRotor();

void SetCountRotor(int countRotor);

};

Rotorcraft.cpp

#include "Rotorcraft.h"

#include "CheckValidation.h"

Rotorcraft::Rotorcraft() : EngineAirTransport()

{

\_countRotor = 4;

}

int Rotorcraft::GetCountRotor()

{

return \_countRotor;

}

void Rotorcraft::SetCountRotor(int countRotor)

{

\_countRotor = CheckValidation::CheckNegative(countRotor);

}

Тесты:

